Spring 1-1-2012

Creating Science Simulations Through Computational Thinking Patterns

Ashok Ram Basawapatna
University of Colorado at Boulder, basawapa@colorado.edu

CREATING SCIENCE SIMULATIONS THROUGH COMPUTATIONAL THINKING PATTERNS

by

ASHOK RAM BASAWAPATNA

B.S., University of Colorado Boulder, 2003

M.S., University Of California Santa Barbara, 2005

M.S., University Of Colorado Boulder, 2009

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirement for the degree of

Doctor of Philosophy

Department of Computer Science

2012

This thesis entitled:
Creating Science Simulations Through Computational Thinking Patterns
written by Ashok Ram Basawapatna
has been approved for the Department of Computer Science

Dr. Alexander Repenning

Dr. Michael Eisenberg

Dr. Michael Klymkowsky

Date

The final copy of this thesis has been examined by the signatories, and we
Find that both the content and the form meet acceptable presentation standards
Of scholarly work in the above mentioned discipline.

IRB protocol #12-0548, 12-0141, #10-0305

Basawapatna, Ashok Ram (Ph.D., Computer Science)

Creating Science Simulations Through Computational Thinking Patterns

Thesis directed by Professor Alexander Repenning

Computational thinking aims to outline fundamental skills from computer science that everyone should learn. As currently defined, with help from the National Science Foundation (NSF), these skills include problem formulation, logically organizing data, automating solutions through algorithmic thinking, and representing data through abstraction. One aim of the NSF is to integrate these and other computational thinking concepts into the classroom.

End-user programming tools offer a unique opportunity to accomplish this goal. An end-user programming tool that allows students with little or no prior experience the ability to create simulations based on phenomena they see in-class could be a first step towards meeting most, if not all, of the above computational thinking goals.

This thesis describes the creation, implementation and initial testing of a programming tool, called the *Simulation Creation Toolkit,* with which users apply high-level agent interactions called *Computational Thinking Patterns* (CTPs) to create simulations. Employing Computational Thinking Patterns obviates lower behavior-level programming and allows users to directly create agent interactions in a simulation by making an analogy with real world phenomena they are trying to represent. Data collected from 21 sixth grade students with no prior programming experience and 45 seventh grade students with minimal programming experience indicates that this is an

effective first step towards enabling students to create simulations in the classroom environment. Furthermore, an analogical reasoning study that looked at how users might apply patterns to create simulations from high-level descriptions with little guidance shows promising results. These initial results indicate that the high level strategy employed by the Simulation Creation Toolkit is a promising strategy towards incorporating Computational Thinking concepts in the classroom environment.

Dedications

To Mom, Dad, and Anand, I love you, thank you for everything.

To all my family, friends, classmates and teachers who have been a part of my life the last 5 years, thank you for always being there for me.

Acknowledgements

I would like to thank,

My advisor Alexander Repenning for encouraging and guiding me through the Ph.D. process and for exposing me to countless educational experiences along the way;

My committee members Alexander Repenning. Clayton Lewis, David Webb, Michael Klymkowsky, and Michael Eisenberg for all their help;

My parents Vara Basawapatna, Ganesh Basawapatna and my brother Anand Basawapatna for all their unwavering support;

Everyone at AgentSheets and my research group who helped me including Fred Gluck, Andri Ioannidou, Kyu Han Koh, and Michael Minerva.

Mark Savignano and Marco Cornacchionne for being incredible teachers who not only helped immensely with this thesis, but also, were wonderful during the GK-12 experience;

And all the students and teachers at Centennial Middle School and Nederland High School and the University of Colorado Scalable Game Design Summer Institute who not only took part in my lessons and gave me valuable feedback over the last 4 years, but also, made this an enjoyable experience.

CONTENTS

CHAPTER

TABLES

FIGURES

CHAPTER I

1. INTRODUCTION

Currently, a major emphasis of many educational end-user programming tools is to empower students with the ability to easily and very quickly program games [1,2,3,58,71]. There are multiple advantages to introducing students to programming through game design rather than more conventional techniques such as C++ and/or Java. For one thing, students tend to like and are engaged by games [4,5,59,80]. Often, these game programming tools are able to eliminate confusing and tedious syntax rules of conventional programming languages by having students program visually instead, for example, with drag and drop rules. This quickly introduces students to the underlying problem-solving logic of programming [6,73,75,76]. Finally, these tools allow students to create 2-D and even 3-D games rather quickly as opposed to having to learn sophisticated graphics packages that accompany traditional programming languages such as C++ and/or Java [6,7].

One such end-user game programming tool, that we currently employ in the Scalable Game Design research group at the University of Colorado, is called AgentSheets and its subsequent 3-D version AgentCubes. AgentSheets/AgentCubes is a rapid agent-based visual game and simulation prototyping environment [8,60,79]. In AgentSheets, middle school students can go from having no prior programming experience to programming their first graphical 2-D game within 5 hours [7]. As students gain expertise, the games they program can become more complex integrating higher-level concepts from both math and science [3].

End-user game programming tools, such as AgentSheets/AgentCubes, have been shown to successfully increase student motivation in I.T. and computer science [4,5]. Though increased motivation is arguably a valuable result of any end-user programming tool, the question arises as to what exactly students are learning from this practice of game design? Many end-user programming tools claim that students learn "computational thinking" [9,10,25]. However, it is not clear what this actually means. At present computational thinking is defined (by the 2011 CSTA, ISTE, NSF standard on computational thinking) as the following [11]:

1) Problem Formulation such that it enables people to use computers and other tools to help solve these problems.
2) Logically organizing and analyzing data
3) Representing data through abstractions such as models and simulations
4) Automating solutions through algorithmic thinking
5) Identifying, analyzing, and implementing possible solutions with the goal of achieving the most efficient and effective combination of steps and resources
6) Generalizing and transferring this problem solving process to a wide variety of problems

Table 1: 2011 CSTA, ISTE, NSF Definition Of Computational Thinking

Even with this definition, it is necessary to concretely describe what computational thinking means in the domain of end-user programming tools. Namely, what skills and/or concepts do we expect students to gain through an end-user programming activity that teaches computational thinking? How do various end-user programming tasks relate to the above definition?

Len Scrogan, former Director of Technology for the Boulder Valley School District (BVSD) in Boulder, Colorado, asked his student an enlightening question as to the actual educational content of a given

AgentSheets project. Mr. Scrogan admitted that he was confused as to the definition of computational thinking, but he went up to a student designing 'Space Invaders' and asked:

> "Now that you've made 'Space Invaders' can you create a science simulation? [10]

This quote is notable for multiple reasons. For one, though this is not a specific list of what students should learn through a given AgentSheets game programming activity, it is a teacher's *expectation* of what a student should be able to achieve after completing this activity. Secondly, the teacher specifically wants to see the student have the ability to create a science simulation using this end-user programming tool. Finally, Mr. Scrogan went on to explain that this was his interpretation of what computational thinking might look like in his classroom. To better understand this third point concerning computational thinking, let us take a moment to consider the concept a little more in-depth.

1.1 Relationship of End-User Game Programming to Computational Thinking

Jeanette Wing, in her article "Computational Thinking", begins to outline why computer science is fundamental and what general concepts from computer science should be considered necessary for problem understanding and solving for all humans [12]; this set of concepts is referred to as "computational thinking." A major component of computational thinking is the ability for students to learn how to utilize computer tools and computational methods to solve problems encountered in everyday life [12,13,14]. Specifically, Jeanette Wing states:

> "Computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science. . .Computational thinking is reformulating a seemingly difficult problem into one we know how to solve, perhaps by reduction, embedding, transformation, or simulation [12]."

For our purposes in game design, the actual game creation gives students the skills necessary to program. Students are 'computational thinking' when they *transfer* these skills learned in game design to using the computer to help solve a real world problem. The above Len Scrogan quote uses the idea of creating a "science simulation" as an example of how the student's skills garnered through programming "Space Invaders" can be applied in a way that would be classified as computational thinking. This idea is underlined by the computational thinking definition above, which emphasize problem solving, abstraction and transfer (see section 1). Ideally, students would take a problem from the real world, represent this problem in the computer domain (in our case AgentSheets/AgentCubes), and do things like running simulation trials and varying parameters to get a better idea of how this "system" might work in different conditions. Students would have to use abstraction to represent the system and this would emphasize conceptualization. In another paper, "Computational Thinking and Thinking about Computing", Jeanette Wing states:

> "In working with rich abstractions, defining the 'right' abstraction is critical. The abstraction process—deciding what details we need to highlight and what details we can ignore—underlies computational thinking [13]."

Furthermore, students might use mathematics or engineering concepts to analyze the data they obtain from a simulation, which is also an aspect of computational thinking [15]. The idea of using simulations as a way

to conceptualize and solve real world problems students encounter is summarized by Figure 1. The representational system in this case is an abstraction that ignores some aspects of the real world system and highlights others in order to solve a particular problem in the real world system.

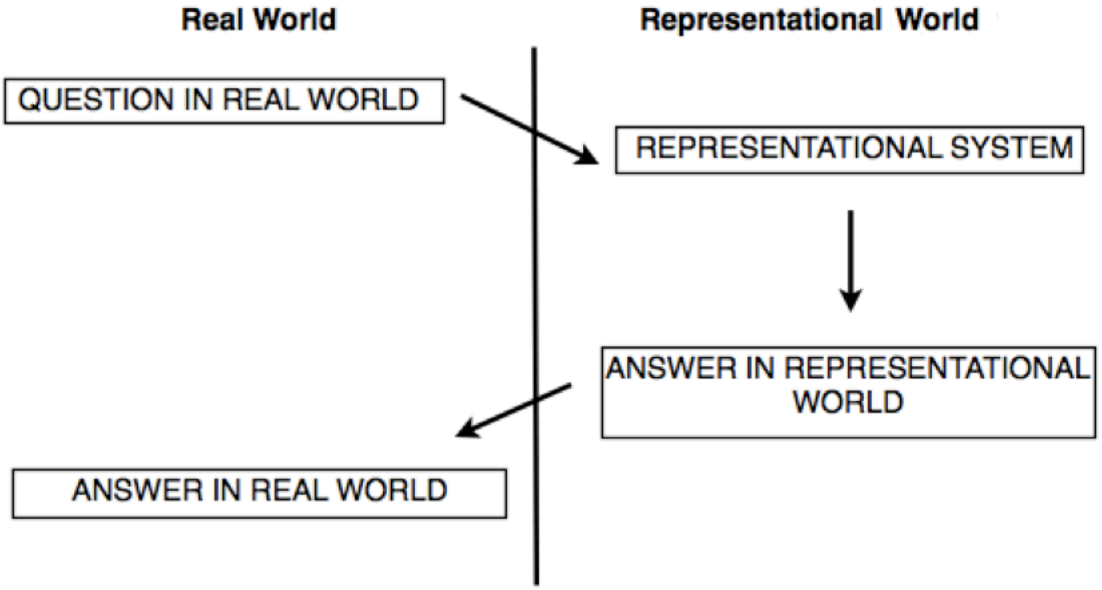


Figure 1: Using A Representational System, Such As A Computational Model, To Solve A Real World Problem[1]

Multiple classes of varying grade levels have successfully used a model of first introducing students to game design using AgentSheets and later, having students create science simulations [7,10,16]. Some of these classes are administered as part of University of Colorado Scalable Game Design National Science Foundation (NSF)-Funded iDREAMS (Integrative Design-based Reform-oriented Educational Approach for Motivating Students) project. The iDREAMS Scalable Game Design Project aims to get a required one week (or more) game design unit in middle school classes of differing socioeconomic profiles, across Colorado and the United States. In

[1] Used with permission from a lecture by Dr. Clayton Lewis, Professor of Computer Science, CU Boulder.

addition to middle school classes, the iDREAMS project includes a two week Summer Institute at the University of Colorado wherein teachers who are planning on teaching these Scalable Game Design units learn how to program in AgentSheets [7].

So far we have only outlined how games enabling science simulation creation could be a valid interpretation of students learning computational thinking. The question remains, what are the elements of game programming that lend itself to computational thinking? To put this another way, what are these units of transfer between games and science simulations? To concretize this concept, we have developed Computational Thinking Patterns.

1.2 An Introduction To Computational Thinking Patterns

One method we have found of abstracting end-user game design is through what we call Computational Thinking Patterns (CTPs). CTPs are design patterns that students initially learn in game design and implementation, but then, transfer to the creation of science simulations [7,10,67,68]. They are groups of behaviors (conditions/actions) given to one or more agents in games that are very similar to groups of behaviors you would use to create agent interactions in science simulations (behaviors are covered more in-depth in section 1.4.1). CTPs can be thought of as the units of transfer between game design and science simulations. Figure 2 depicts the Computational Thinking Pattern Spiral which informally shows the flow of patterns from basic to more sophisticated and how they relate to different mechanisms in game design and science.

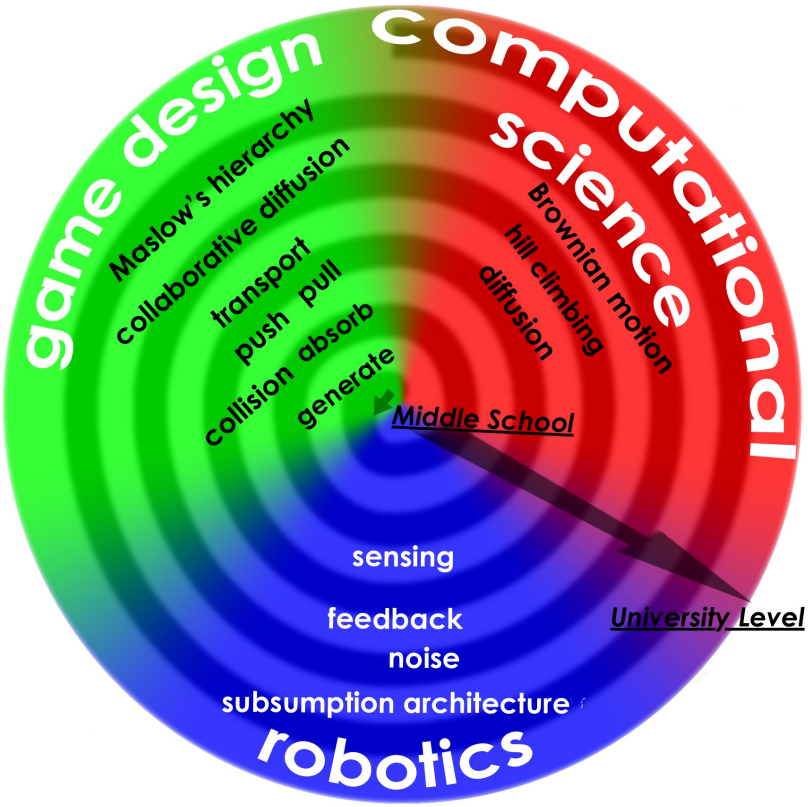


Figure 2: Computational Thinking Spiral depicts Computational Thinking Patterns from basic ("Middle School") to advanced ("University Level") [7].

To get a better understanding of Computational Thinking Patterns, it is helpful to look at the following illustrative examples of patterns that occur in games and simulations- these can also be found in [7]:

Generation: This pattern occurs when one agent creates one or a flow of other agents; examples are found in the game Frogger, where tunnels create trucks at the beginning of the road once every so often. In science simulations, such as predator prey simulations, generation is used among agents to create offspring.

Absorption: The inverse of generation, absorption occurs when one agent makes another agent disappear. This occurs in games like space invaders

when a bullet "absorbs" aliens (then erases itself). In science simulations, when a predator eats a prey it absorbs it.

Collision: Collision occurs when two agents physically collide with some consequence. Absorption can be thought of as a type of collision for example. Other examples can be found in Frogger when the frog collides with a truck yielding a dead frog or artillery when a cannonball collides with an opponent's base.

Transportation: In this pattern, one agent carries another agent. In Frogger, the frog gets transported by a log. In science simulations, one can think of oxygen being carried by red blood cells.

Push: The push pattern occurs when one agent pushes another agent to another location in the world. This occurs in the Sokoban game when an agent pushes a box onto target locations.

Pull: Similar to push, the pull pattern occurs when one agent pulls another agent to another location in the level.

Diffusion: This pattern occurs when an agent's "scent" is spread around a level. The location of the agent at a particular moment is where the scent is the highest. Some sort of background agent helps to diffuse the scent through the world as dictated by an equation that ensures the scent value decreases as the distance from the agent, where the scent is emanating from, increases. This occurs in "Pacman" when Pacman himself has his scent spread around the level. In science simulations a light bulb might diffuse heat and light out into the world.

Hill Climbing: Hill climbing allows agents to automatically follow a diffused scent (among other things) and, thus, enables one agent to track another agent. In Pacman, certain ghosts could use hill climbing to actively chase and

track Pacman's scent. In science, a bee might track the diffused scent of a flower or a snake might track the heat scent of a rat.

The above list is by no means exhaustive and finding and discovering new Computational Thinking Patterns is a work in progress. In fact, making students program science simulations by using Computational Thinking Patterns might give insight into other patterns that are necessary to define. For example, to implement interactions similar to how an enzyme might break down a carbohydrate chain we might introduce a *Link* and *Cut* pattern so that our glucose agents can link together to form carbohydrate chains and our enzyme agent can cut these links to free glucose. These patterns give us an initial insight into how skills learned in game programming might readily be used in the creation of science simulations.

We have defined Computational Thinking Patterns as the units of transfer between game programming and science simulations. One might wonder whether it would be useful to program at this Computational Thinking Pattern level itself. For example, let us say that a student studied a given interaction in science class. At this point we might give the student a palette consisting of animated icons acting out various Computational Thinking Patterns and ask the student to combine patterns together, and specify the acting agents, to create a simulation of the interaction they just learned, and experiment on it. This begs the following big picture question: can we use the idea of Computational Thinking Patterns as an abstraction that lowers the programming barrier enabling students to create science simulations by directly using the analogy between the science simulations and the animated agent interactions they see? In other words, instead of students implementing these patterns, can students utilize these high-level patterns directly, as the elemental units of programming, in order to create

simulations quickly and to better understand the concepts they are learning in the classroom? If this could indeed occur, such a tool wherein students programmed by exclusively using agent interactions as defined by Computational Thinking Patterns could be a powerful tool for classroom integration of computer science and modeling concepts as well as increasing student understanding on a particular topic.

Even if we could create such a system the question remains what possible advantages would such a high-level system have over traditional end-user programming tools in a classroom setting? The remainder of this introduction will aim to answer this question

1.3 Current State of Modeling And Simulation Creation In The Classroom

1.3.1 GK-12 ECSITE Experience.

From Summer 2009- Summer 2011, I was part of the National Science Foundation GK-12 ECSITE (Enabling Computer Science In Traditional Education) program run by the Department of Computer Science at the University of Colorado, Boulder. The ECSITE GK-12 fellowship aims to use graduate students to teach STEM education in middle and high school classrooms as well as incorporating computer science and Computational Thinking into the traditional curricula of K-12 schools. In my personal experience interacting with students over one and half years in classrooms at Centenary Middle School[2] (Boulder Valley School District, Boulder, CO.), Neds High School (Boulder Valley School District, Boulder, CO.), and CABPES (Colorado Association of Black Professional Engineers and Scientists- a non-profit dedicated to improving underrepresented minority

[2] Names of teachers and schools have, for the most part, been changed in this thesis.

numbers in the maths and sciences in Denver, CO.), I have noticed much of the computer-related activities occur in computer class exclusively. Often, these computer classes are electives and, thus, the general school populace is not required to take these classes. Furthermore, because these classes are disconnected from other classes in the curriculum, students do not necessarily learn how computers could enable better understanding of and help in solving problems encountered among a variety of different disciplines.

For example, sixth grade students at Centenary Middle School are only required to take one eight-week quarter of computer-related activities as part of an "exploratory wheel" wherein they do multiple electives throughout the year. Of this eight-week course, one week is dedicated to end-user game design programming using AgentSheets. The remainder of the course is spent doing non-programming related tasks like Flash animation, Photoshop, PowerPoint, and Comic-Life. After this class, students are only required to take one more semester's worth of computer related classes or electives again for the remainder of their middle/high school career. These classes may or may not involve subsequent AgentSheets programming (some might emphasize Photoshop again or Flash animation). Some Centenary students end up in Neds High School, which, like all Boulder Valley School District (B.V.S.D) high schools, does not require a Computer-related elective if one was taken in middle school. Therefore, in the seven years from sixth grade until graduation, a student may only have 1-2 weeks of actual end-user programming experience. Furthermore, a student can graduate from high school without ever being exposed to simulation creation and modeling.

In general, B.V.S.D, within which Centenary Middle School and Neds High School resides, is considered by many to be a technologically advanced district; so much so, in fact, that the former Director of I.T. for BVSD, Len

Scrogan, was given his own TED talk [18]. Similarly, through my experience teaching at CABPES, the students, who come from public schools across the Denver metro area, state that they learn things like AgentSheets, Scratch and Alice in a computer class or, sometimes, at an after-school computer club if they learn it at all.

Though enabling users to design and implement games in a computer class/club environment is helpful for motivating and even teaching students in computer science and technology for the future, it is not necessarily the best application of computational thinking. There are two reasons this is the case. The first is that students are not using their computational problem solving experience outside their limited exposure in a narrow context among other computer related activities. Second, though motivational, this experience introduces the question of what is actually being learned? Again, we can go back to Mr. Scrogan's quote above wherein he asks how does learning game programming enable users to actually problem solve. If indeed these end-user programming tasks actually give users problem solving skills should not students be able to use and hone these computational skills in an actual science class solving problems and gaining understanding of concepts they naturally encounter in their respective curricula? Why is this not happening?

As part of my work, I interviewed Marco Cornacine (7th grade Life Science, Centenary, Colorado) Will Underjohn (7th grade Life Science, Centenary, Colorado). All the Life Sciences related quotes I use are from Mr. Cornacine because this is Mr. Johnson's first semester in teaching Life Sciences as opposed to Mr. Cornacine's 17th year; however, Mr. Johnson supported everything Mr. Cornacine said. In my interactions at Centenary Middle School, teachers point to the fact that doing a computer unit in a non-

computer class is often not possible as a computer unit must be tailored to the current lesson plan. For example, when I asked Mr. Cornacine the biggest barriers to integrating a computer modeling exercise into the class curriculum he said:

> "Time to put together the resources, finding activities that are appropriate."

However, he went onto state that the few times he had done computer related activities in class he had found that it totally supported the Guaranteed Viable Curriculum of Centenary. Oftentimes, programming is out of the question because the overhead of having students learn (or re-learn) how to program and then create a science simulation, for example, is prohibitive given the curriculum and time constraints on the class. When I asked Mr. Cornacine if he had ever done a programming modeling unit in his Life Science class he said:

> "Never done a programming (modeling) unit in any of the Life science classes. . .unfortunately (because) that's the logical application."

He went onto say that a few more simulations on larger processes like cellular respiration or at the macro level, like natural selection, would be perfect because these units are harder for kids to visualize and they are not conducive to lab experiments. Mr. Cornacine also said that he could get into the computer lab once every two weeks if he needed to, but usually uses it once a semester. Mr. Cornacine likes integrating computers into Life Science classes, believes that it helps fulfill his curriculum requirements and even has access to computers—however, the overhead of integrating a

programming modeling unit, though appealing, is too much in terms of lesson development.

1.3.2 My Experience With Simulations In The Classroom.

In addition to the above general experiences, I had four enlightening Life Sciences experiences, three during my time in GK-12 and one as an instructor for the University of Colorado Science Discovery summer class. The first experience occurred when I went to Centenary and helped Marco Cornacine and one of his students set up a lab for students that dealt with the enzyme Amylase in the Fall 2010 semester. Amylase is found in human saliva and helps break down starch into glucose. This lab involved students putting starch, water, and saliva into a test-tube and then testing for the presence of glucose using Benedict's solution. After I helped set up the lab, Marco asked if I could go online and find an animation of the Amylase enzyme "cutting" the starch molecule into individual glucose molecules. After a cursory search, I found a flash animation that depicted this interaction and Marco got excited stating that the animation would be great in helping students better visualize what is 'actually happening'[19]. The above lab treats the actual enzyme interaction as a black box. The only reason we know that starch is being broken down into sugar is because we are told that Benedict's solution will turn a certain color in the presence of glucose and we test the reaction output of saliva and starch using this test. The animation was shown in all eight Life Science classes across all three teachers in Centenary seventh grade Life Science.

Given that the current lab is such that an animation is beneficial for student knowledge, how much more could be learned if these students could actually program this interaction? This could not only lead to a visualization

that the students create themselves, but also, lend itself to valuable further experimentation. For example, students could see the effects of actually varying the amount of amylase in the saliva and the effects of breaking down starch yielding valuable conclusions as to energy production or, if time permits, motivate even greater findings. An example of why this might be important comes from the *Wikipedia* entry on amylase wherein it states (citing a Scientific American article from the April 2010 issue):

> "Amylase is thought to have played a key role in human evolution in allowing humans an alternative to fruit and protein. A duplication of the pancreatic amylase gene developed independently in humans and rodents, further suggesting its importance. The salivary amylase levels found in the human lineage are six to eight times higher in humans than in chimpanzees, which are mostly fruit eaters and ingest little starch relative to humans. [20,21]"

Students could vary the amylase level in a simulation and actually see why it might be infeasible for chimpanzees, with less concentration of amylase in their saliva, to digest as much carbohydrates. This could motivate the differences in diets between chimpanzees and humans as well as introduce students to macro evolutionary behavior on a molecular level.

It should be noted that in the Fall 2011, a semester after I had completed my GK-12 fellowship, Marco asked me if I still wanted to build an amylase simulation for his Life Science Class. I created the simulation in AgentSheets and we taught it over two days in all five of his classes. No data was collected from this unit, however, Marco used this unit to introduce the idea of experimenting on simulations. We had the students experiment with various amylase saliva concentrations, using the chimpanzee amylase concentration as motivation, to see how long it would take to break down all the starch in a given trial run. Furthermore, we had the students calculate

how much energy was produced in a given simulation run and equate it to the energy it would take a given sized human to perform everyday tasks (i.e. like climbing the stairs etc.). Afterwards, Marco said that this was a great way to introduce the ideas of food and energy to students.

The second Centenary experience occurred in Spring 2011 when I taught a Life Sciences epidemiology-related unit through GK12. I created an AgentSheets "Epidemiology Simulation" (Figure 3) to help students visualize and interact with the spread of diseases, and specifically, investigate ideas like transmission rate on varying age groups populations as well as using models to develop a real-world strategy. Ideally, students would have programmed the epidemiology simulation themselves. However, due to curriculum time constraints we only had two days in the computer lab; therefore, instead of having students program the simulation, they just ran experiments on a pre-made simulation. I had students use the simulation, develop and inoculation strategy, and analyze the assumptions made in the model.

Students were given a population consisting of 3 different agents: Elders, Adults, and Kids who all move, contract the virus, become healthy, and die at different rates. There were 200 agents in approximately the same ratio to one another as the United States population (50 kids- ages under 18, 125 Adults- ages 18 to 65, and 25 elders- ages greater than 65). Students were first told to run the simulation four times and to calculate the average dead of the four trial runs without distributing any inoculations. Then, students were given 40 inoculations to spread among the populations in any way they chose. For each 'inoculation strategy' students would similarly run the trial four times and average the total dead results. Students were told to run 3 to 4 inoculation strategies and talk amongst themselves to learn which

strategies seemed to work the best before formulating subsequent strategies. This was run among eight different seventh grade Life Sciences classes over 16 class periods.

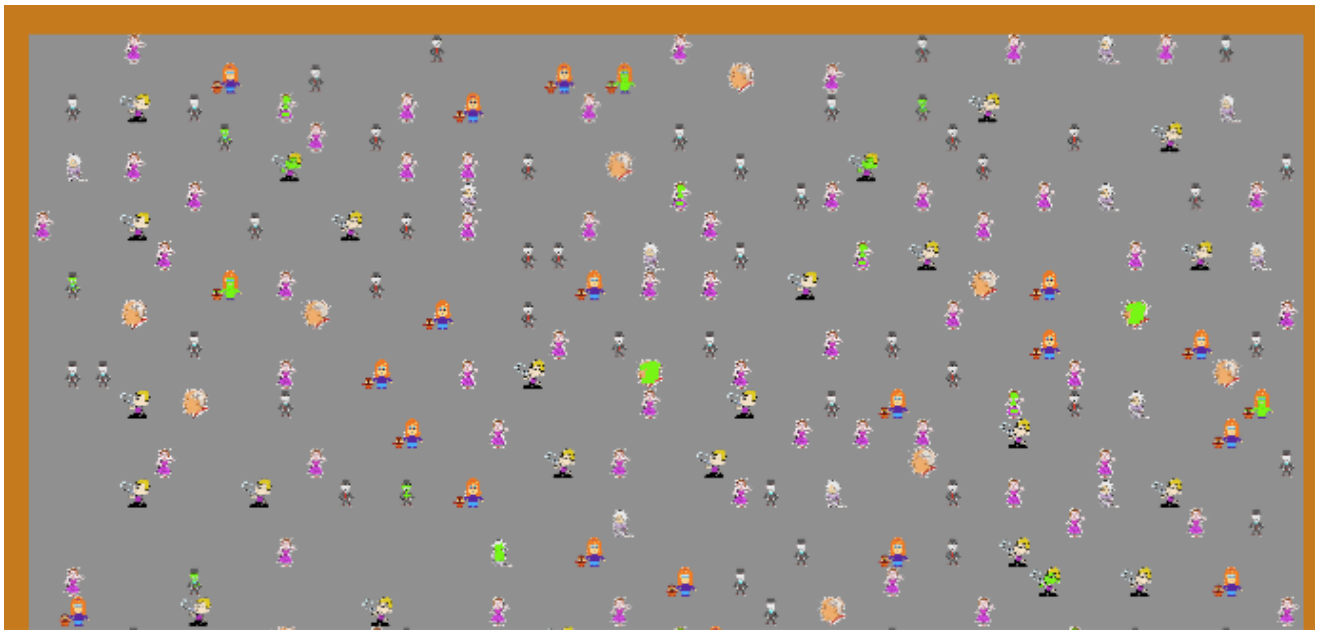


Figure 3: Screenshot Of Epidemiology Simulation Used At Centenary Middle School. Six Agents Are Pictured, Male/Female Kids, Adults, and Elders. The Green Agents Are Sick.

After the unit was taught to all the classes, the teacher-reaction was overwhelmingly positive. One teacher, Bill Schmoker, emailed me saying that the children learned a lot from it and used the data to prompt further research into how diseases spread. Another teacher, Will Underjohn, asked that I come up with other areas where we can integrate computation into the classroom and further stated that he thought the unit helped students learn a lot more about simulations and diseases themselves than they would have otherwise. From this experience two things are clear:

1. The Life Science teachers at Centenary have shown an extreme interest in integrating a computer aspect to their current units.
2. According to the teachers, the computer-based lessons, though short in timespan, gave students an initial look at how they could use computer science as a method for problem solving.

Keep in mind that this lesson only had students alter simulation parameters--one wonders how much more students could hypothetically learn if they actually programmed and then altered their own simulation? The days prior to me teaching this unit, each student was given a different disease to research and do a final presentation on at unit's end. How would a student's understanding of their disease improve if they could have modeled the disease they were researching? In programming the simulation students would learn how to choose parameters that go into model creation and be able to do things such as model different diseases as well as different inoculation strategies. They could even analyze in what situations their model was effective and in what situations their model oversimplified real life. I think we could argue that this and other activities, integrated into the curriculum, would be an important contribution to computational thinking within the classroom.

The third experience happened during my first year in the GK-12 program. I was working in Marks Savs's Neds High School computer class. As part of our AgentSheets programming unit, I helped teach the kids how to create a Predator/Prey simulation. For this particular simulation we used ball python and rat agents. The ball python agents would get hungry, chase rats using an equation taken from [22], and if it encountered one, would eat it. If it did not encounter a rat in time it would die. Both agents could create more agents through mating. Though this seems like a simple simulation to

create, it took a week for many of these high school students to complete. One could imagine that in middle school this activity would take much longer. Furthermore, the idea of integrating three or four such units or integrating a more complicated simulation in a middle school classroom setting would take too much time.

The fourth experience occurred when I was an instructor at CU's Science Discovery summer camp for kids. During the last two weeks of summer, Science Discovery runs a class called iCAMP wherein students, ages 8-14, are able to work on multiple technology related projects. One of the projects students could choose to do was to make a game using Game Maker Pro, an end-user video game software creation tool. Many students were initially excited by the idea but were quickly frustrated with how hard and tedious it was to make a game using this tool and would give up.

I noticed that the computers had a Microsoft Research end-user game development tool called Kodu[3] preloaded on them. I inquired with the iCAMP personnel and they said no instructor had used it yet but I should feel free to try it out with my class. I had been exposed to Kodu before at various conferences and had played around with it on my own, and though it is far from perfect, it enables, to a certain extent, students to program high level behaviors on their in-game characters. After introducing this to students, most students immediately switched from using Game Maker Pro to Kodu quickly creating a variety of games. Though this is a single example over two one-week classes, it illustrates the power of high level programming to lower the accessibility barrier for first-time users and enable more rapid game creation for all users.

[3] Research.microsoft.com/en-us/projects/kodu

Given these four experiences, one wonders if combining a high-level tool to create simulations might be a first step towards enabling classroom integration of computational thinking concepts?

1.3.3 The Current State Of Computation In The Classroom Nationwide

It should further be noted that, though these above experiences are limited in scope, this trend is echoed on a large scale across the United States. The 2009 U.S. Department of Education Report entitled *"Teachers' Use of Educational Technology in U.S. Public Schools"* surveyed teachers to find out to what extent technology is integrated into U.S. classrooms [23]. Table 2 shows excerpts from this study for four different questions and two different types of class categories: 'math/cs/science' classes and 'general education in self contained classroom' classes. Teachers could answer 'never', 'rarely', 'sometimes' or 'often' to each question; the table in the report (and similarly in Table 2) combines 'sometimes' and 'often' answers and excludes 'never' answers.

	Simulation and Visualization Programs		**Solve problems, analyze data, or perform calculations**		**Conduct experiments or perform measurements**		**Develop or run demonstrations, models, or simulations**	
	Rarely	Sometimes or often	Rarely	Sometimes or often	Rarely	Sometimes or often	Rarely	Sometimes or often
math/cs/science classes	24	45	20	61	27	45	27	25
General education in self contained classroom	24	30	19	43	24	18	17	10

Table 2: Excerpts US Dept. Of Education NCES Study Recording Percent Of Teachers Reporting How Frequently Their Students Performed Various Activities Using Educational Technology During Their Classes

We see from this table that math/cs/science classes consistently use

simulation and visualization programs, do problem solving, conduct experiments or perform measurements, and develop or run demonstrations, models, or simulations more than general education classes in a self contained classroom. This should come as no surprise; however, what is startling is how low math/cs/science classes are in these categories. For example, in terms of "developing or running demonstrations, models, or simulations", the category in which any AgentSheets/AgentCubes Scalable Game Design activity would fall into, only a quarter of the classes do this often and almost half of these classes never partake in this. Many of these math, science, and C.S. classes rarely or never employ "simulations or visualization programs", a key use of computers in the classroom. Given that computer science is grouped with math and science classes, one would expect that if we had percentages exclusively for math and science classes in these categories, they would be lower.

The National Science Foundation echoes the above study results and Marco Cornacine quote (see above section 1.3.1) in a press release stating that:

> ". . .most middle and high school students receive no exposure to computer science. One major obstacle to educating young students in computer science is finding a space for a computer science class in an already overburdened K-12 curriculum [24]."

Furthermore, another National Science Foundation article talks about how the lack of Computational Thinking into classrooms could make students less competitive in the long run.

> "As the 21st Century world becomes more dependent on the skills gained from computer science, like complex problem solving and analytical thinking, the lack of computer science knowledge may put our young people at a disadvantage [25]."

The Common Core State Standards Initiative coordinated by the Governors Association Center for Best Practices and the Council of Chief State School Officers wrote middle and high school standards based on 10,000 responses from multiple national organizations. Among their standards is a high school standard for Mathematical Modeling. Their example models include *"Analyzing risk in situations such as extreme sports, pandemics, and terrorism"* which is similar to the above epidemiology example [25]. Judging from the above quotes by the NSF and Mr. Cornacineichione, as well as NSF funding towards after school computer clubs, summer camps and other computer enrichment activities, Presidential Advisory Committee endorsements [74], not to mention third party non-profit foundations, such as the SHODOR foundation[4], that specialize in getting computation into the classroom, there is a strong perceived need for overcoming these challenges and exposing middle school students to Computational Thinking and Computational Science. It is my hope that this project can be step towards fulfilling this need.

1.4 An Introduction To The Simulation Creation Toolkit

This thesis aims to investigate a first step in fulfilling the need for simulation and modeling creation activities in the classroom. To achieve this, a tool called the Simulation Creation Toolkit has been created. The Simulation Creation Toolkit allows users to create interactions between agents at a higher level, namely, the Computational Thinking Pattern level (see section 1.2). The hope is that allowing students to program at this level will enable students to quickly create simulations by making an analogy with

[4] Shodor.org

particular science phenomena they are trying to simulate. Let us take a closer look as to the differences between programming in AgentCubes and the Simulation Creation Toolkit to get a better idea as to why this strategy is worth investigating.

1.4.1 Brief Introduction To AgentCubes

What follows is a very cursory introduction to AgentCubes. This will cover just enough for the reader to understand the thesis project (with other more complex AgentCubes topics being explained in later chapters when relevant)[5].

As mentioned above, AgentCubes is a rapid game and simulation prototyping environment. The game or simulation characters in AgentCubes are called "agents." Each agent consists of two things: one or many "shape(s)" that determines how it looks and "behaviors" which determine how an agent acts. It should be noted that for our purposes the terms "shapes" and "depictions" can be used interchangeably. The following picture is an example of an AgentCubes agent named "Lobster".

[5] For more information on AgentCubes please go to http://www.agentsheets.com/ and http://scalablegamedesign.cs.colorado.edu/wiki/Scalable_Game_Design_wiki

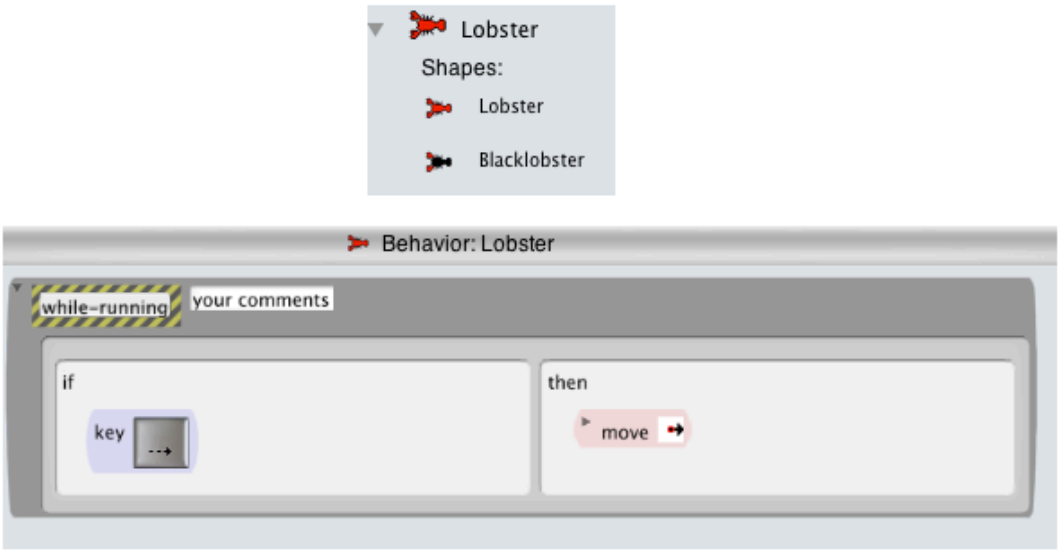


Figure 4: Lobster Agent With Shapes And Behaviors

This Lobster agent has two shapes-- one named "Lobster" and one named "Blacklobster". The creating and editing of 3-D shapes (called "inflatable icons") is an interesting topic but will not be covered in this paper, but information on it can be found at [29]. This Lobster agent has one behavior—if a user hits the right arrow key, the Lobster agent moves to the right. Let us look at the behavior of this agent a little more closely.

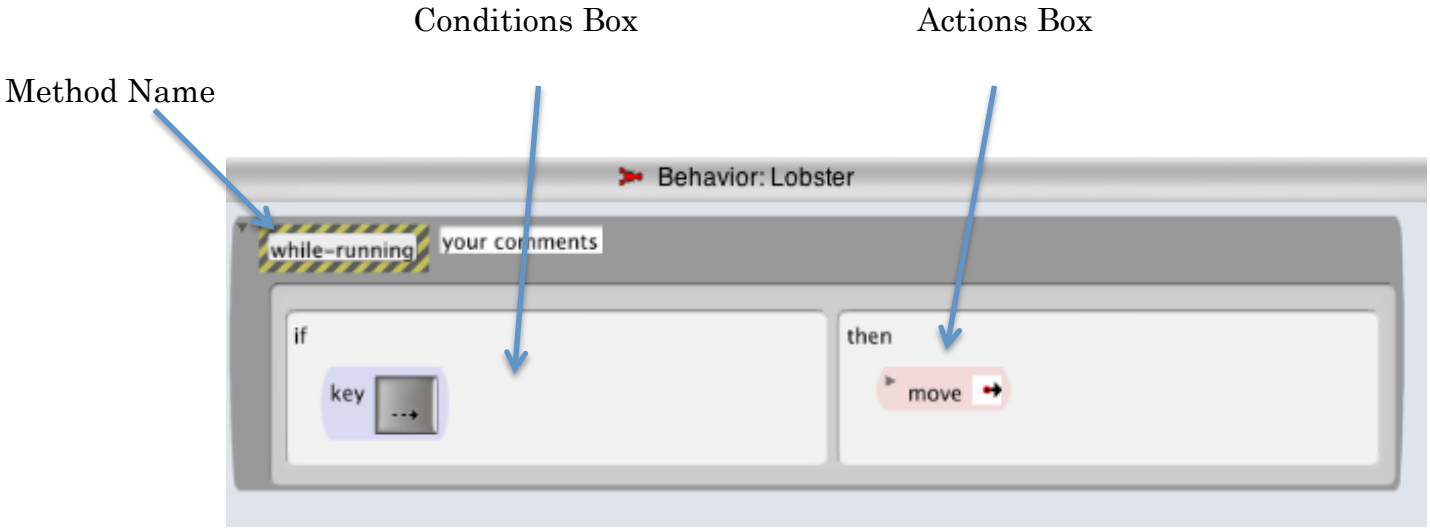


Figure 5: Behavior Of Lobster Agent

Among other things, the above figure depicts the method that this Lobster agent behavior resides. Agent behaviors consist of one or many methods. This particular method, called "while-running", runs in a loop as the program executes. We will also refer to one time through the while running loop of an agent as an "update cycle" for that particular agent. Each method contains one or more "rules". In this case, our while running method has one rule. Each rule consists of an "If" condition and a "Then" action box. The idea is that if the condition is met, then the corresponding action is triggered and no other rule for that method is triggered for that particular agent's update cycle. The rules are checked vertically, thus, earlier rules have a higher priority. The condition box in Figure 5 has a "key" condition. The "key" condition is met when a person hits a particular keyboard key. In this case, the condition is met when a person hits the right arrow key. When a condition is met for a rule, that rule's action is triggered. The action that will be triggered when the right arrow is hit is a "move" action. Specifically, the Lobster agent will move to the right. A condition box can have one or more conditions and an action box can have one or more actions. Multiple conditions in a condition box means that all the conditions must be met for the action to be triggered; similarly multiple actions in an action box means that if that rule's condition(s) are met the agent will act out all those actions. A summary of all the AgentCubes Conditions and Actions can be found in Appendix A.

Finally, the level where the agents act out their behaviors is called the "world". Each world is organized into a grid of squares. Each agent in the world inhabits a square, however, more than one agent can inhabit a square at once (i.e. agents can be "stacked" upon one another). The following figure

depicts a zoomed-out and zoomed-in version of a 20 by 20 AgentCubes world with a Ladybug agent on it.

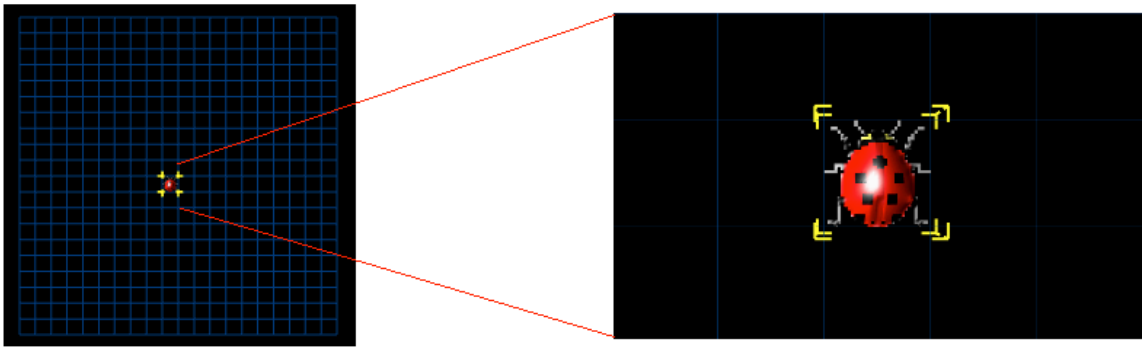


Figure 6: Left Picture is a 20 Grid Square By 20 Grid Square World With Ladybug Agent On It. Right Picture Is A Zoomed In Picture Of The 3 by 4 Area Around The Ladybug Agent

Notice that in Figure 5 we saw that the behavior for the Lobster agent was such that if the right keyboard key is hit the Lobster agent would move right. By this we mean the Lobster agent would move 1 world grid space to the right. Now that we know how behaviors are added to agents let us look at groups of behaviors.

1.4.2 Link Between AgentCube Rules and Computational Thinking Patterns

Section 1.2 introduced Computational Thinking Patterns as the "units of transfer" between games and science simulations. Specifically in AgentCubes, we can say that Computational Thinking Patterns are sets of rules that enable agent-interactions in AgentCubes games and, similarly, those same sets of rules can be used for an AgentCubes science simulation. It should be noted that Computational Thinking Patterns are not limited to AgentCubes and one could come up with implementations of Computational Thinking Patterns using other end-user programming tools. Computational

Thinking Patterns are general descriptions of agent interactions, there are multiple ways of actually implementing each CTP as AgentCubes behaviors. To make this more clear let us look at an example. Say a user programs the behavior in their game for a tunnel to create trucks (as would happen in the game "Frogger"). This pattern is called "Generation", as in the Tunnel agent is generating the Truck agents. The following figure depicts one way a user might program this in AgentCubes.

Figure 7: Behavior For Tunnel Generating A Truck

This above rule, which resides in the Tunnel agent behaviors, states that once every second a new Truck agent will be created to the right. A user might not want the Truck agent to be created by the Tunnel agent every second, but rather, have Truck agents be created with a percent chance. The user might program this behavior as follows.

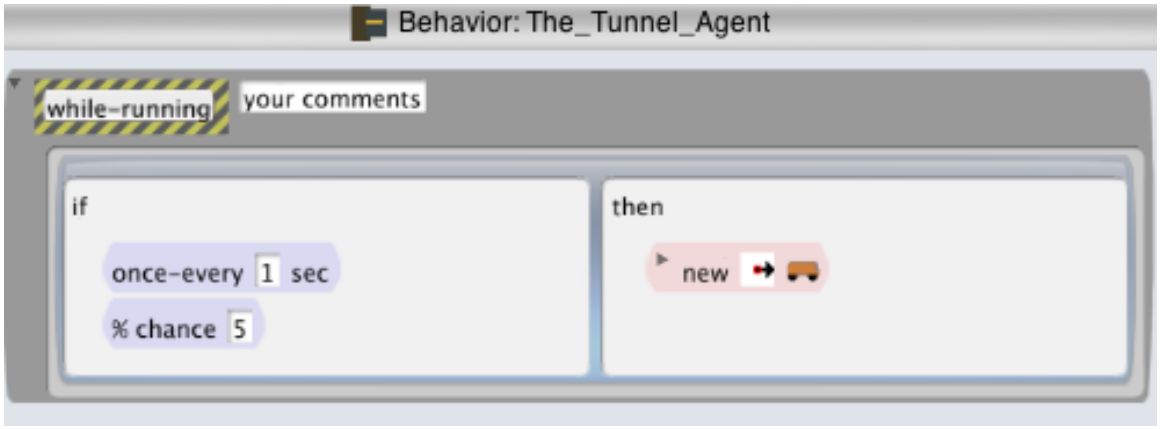


Figure 8: Behavior For Tunnel Generating Truck With A Percent Chance

Figure 8 says that once every second there is a 50% chance that truck will be created to the right of the tunnel. Both Figure 7 and Figure 8 would be considered a type of the Generation pattern because they are both creating a truck as their action.

The implementation of different patterns in AgentCubes varies in complexity, and each pattern itself has numerous ways it could be implemented. The ramifications of this idea will be talked about more throughout this thesis, but for now, let us look at a specific example that highlights a possible benefit of programming at the Computational Thinking Pattern level.

1.4.3 Illustrative Example: Implementing One Agent Tracking Another Agent in AgentCubes

Let us now explore the difference between programming at the AgentCube behavior level and the Computational Thinking Pattern level. One of the patterns introduced earlier is the Tracking pattern (see section 1.2). This pattern occurs when one agent chases another agent around the world. In the above-described predator/prey simulation for example (see section 1.3.2) the Ball-python agent chased the Rat agent when hungry.

From a high level, one way a person might accomplish this is by using the idea of "scent diffusion". Scent diffusion uses the idea of "anti-objects" to carry the scent from the chased agent to the chaser agent [30]. Namely one or more background agents are employed to carry the scent around the world. The agent that is being chased would set a local variable (i.e. local to that particular agent) that represents its scent to a certain value. The background agents would then set their value for this scent to an average of the scent values around them. This would create a scent gradient on the background agents going from the highest scent, which would occur at the location of the chased agent, to the lowest scent, which would occur at the world background agent furthest from the chased agent. Finally the chaser agent would track the chased agent by following this scent. One way to implement this is every time the chaser agent makes a movement, that agent checks the scent of all its surrounding squares and moves in the direction wherein the scent is the highest.

In AgentCubes the behaviors might look as follows. We will use a Ball-python agent as the chaser agent, a Rat agent as the chased agent, and a Tile agent as the background agent. First off the Rat, agent has to set a certain variable representing its scent to a given value. The following figure shows one way this behavior could be accomplished.

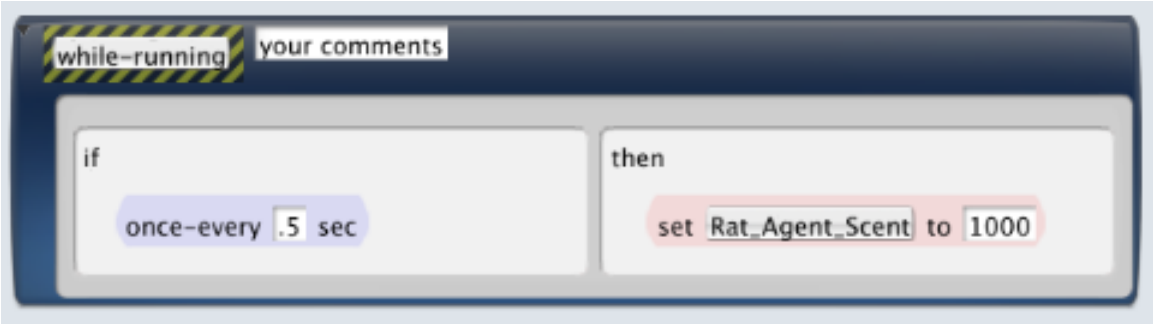


Figure 9: Behavior For Rat Setting Variable "Rat_Agent_Scent" To 1000

The above figure shows the Rat agent's "while-running" method with one condition and one action. The rule states that once every half-second the Rat agent sets a variable called "Rat_Agent_Scent" to 1000.

The next step is to have our Tile background agent diffuse the scent. Recall that each background square averages the Rat_Agent_Scent of the four squares around it to get its own Rat_Agent_Scent value. Therefore the squares around the Rat agent will have higher Rat_Agent_Scent values than squares further away from the Rat agent. The following figure depicts one way we implement this behavior for the Tile agent

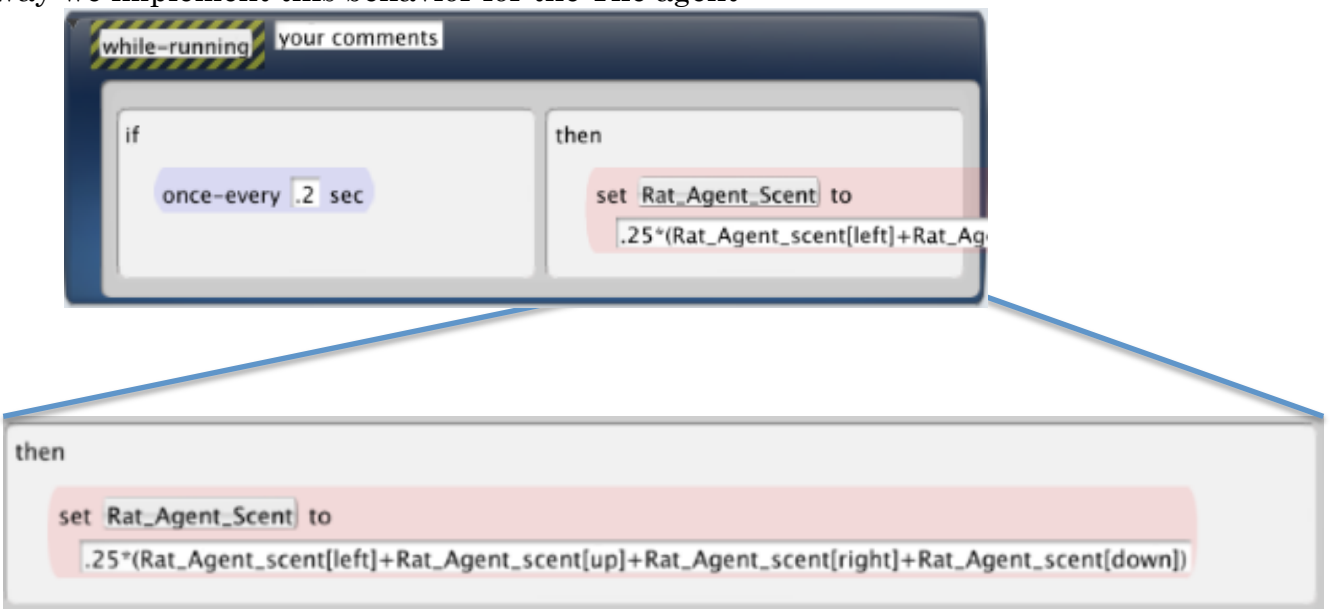


Figure 10: The Top Picture Is The Tile Agent Rule For Diffusing Scent. The Bottom Picture Is The Then Box Blown Up To Show The Whole Diffusion Equation Used.

The top picture in Figure 10 depicts a rule wherein once every .2 seconds the Rat_Agent_Scent variable of the Tile agent is set to something. The bottom picture shows the whole equation that the Rat_Agent_Scent variable is set to: an average of the Rat_Agent_Scent of the four agents surrounding the tile. Two things of note: a user can also do this for the 8 agents surrounding a tile, for now we will look at 4-direction diffusion;

another thing to note, if there are multiple agents on a grid square the Rat_Agent_Scent of the top-most agent is the value used for this equation (if that agent happens to not have a Rat_Agent_Scent variable then its value is zero for that direction). Notice that in order for students to have the background diffuse the Rat_Agent_Scent it takes them typing out an onerous equation wherein mistakes are easily possible. Furthermore it is debatable as to whether a given student actually understands what this, or similar diffusion equations, means.

Finally, our Ball-python agent has to chase the Rat agent by moving only on the square around it with the highest Rat_Scent_Value. Thus we have to add this to the Ball-python behaviors. This is done in another method that is called from the while-running method once every so often. We implement it this way for a variety of reasons that we will not go over now. For our purposes, just assume this method is being called once every half second. The following figure depicts this method that enables the Ball-python agent to track the Rat agent.

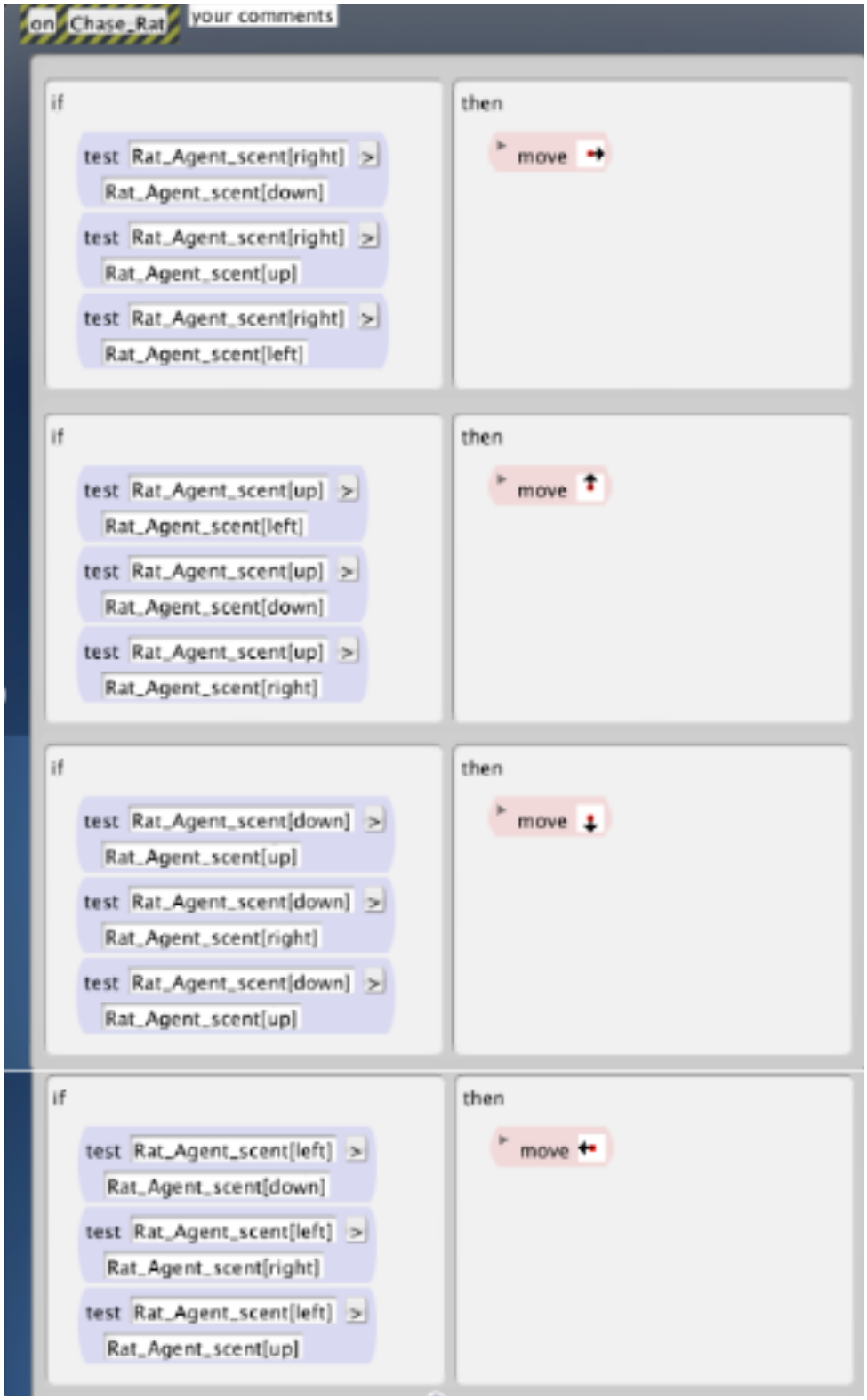


Figure 11: The Method That Allows The Ball Python Agent To Chase The Rat Agent

Figure 11 depicts one method with 4 rules; however, each rule has multiple conditions. Each rule states that if the Rat_Agent_Scent in a given direction is greater than the Rat_Agent_Scent in the other three directions, the the Ball-python Agent will move in that given direction. Again, notice how long these rules are and how easy it would be for a student to make a mistake creating these rules.

The idea behind tracking is fairly straight forward; however, middle school students can often have problems implementing all these rules among the three different agents in AgentCubes. Furthermore, if a student wanted to have another agent also chase the Rat agent, the student would have to add all the rules from Figure 11 to a new agent. Moreover, if a student wanted to have another diffusion in their simulation, they would have to redo all the above rules for a new agent and a new agent scent variable.

The implementation of the Tracking pattern is a wonderful example of an exercise that introduces students to topics outside of merely game programming. For example students begin to learn about "hill climbing" algorithms which is something that they might be able to transfer to their math classes in terms of differential equations. They can play with the Tile agent diffusion equation to see where the level scent values go unstable. However, if the aim of the lesson plan is not to introduce students to these topics, but rather, have them create a simulation and explore those interactions, the implementation of the Tracking pattern can be extremely time consuming and can add unnecessary complexity to a lesson. This was exactly the situation Marks Savs and I found ourselves in when we did the predator/prey simulation at Neds High School (see section 1.3.2). High school students had trouble with the implementation of the Ball-python tracking the Rat agent. Fortunately, those kids were in a computer class wherein we could

spend time exploring the idea of agents diffusing out a scent value and another agent hill climbing towards that scent value. In a middle school Life Science class, there is simply not enough time to explain these concepts everytime they come up for various patterns. Even if a student knew that the predator agent, for example, should at some point chase the prey agent, it is very unlikely that the student would automatically come up with the above implementation solution to this problem using AgentCubes conditions and actions. Now let us look at how this same interaction might be implemented at the Computational Thinking Pattern level in AgentCubes.

1.4.4 Illustrative Example: Implementing One Agent Tracking Another Agent At The Computational Thinking Pattern Level

At the Computational Thinking Pattern level a user no longer has to add each individual condition and action to a given behavior rule. The user simply selects what she/he wants each agent to do and the rules are added automatically behind the scenes. Therefore, from a user perspective, adding behaviors at the Computational Thinking Pattern level could be a lot easier and quicker.

For our purposes in AgentCubes we will use the Simulation Creation Toolkit, developed for this thesis, to implement the same pattern we did in section 1.4.2, namely have the Ball-python agent track the Rat agent. This will be a brief example of the Simulation Creation Toolkit with more explanation of the tool in later sections. For now, say we had the following Tracking Pattern palette that we were allowed to use in our AgentCubes project.

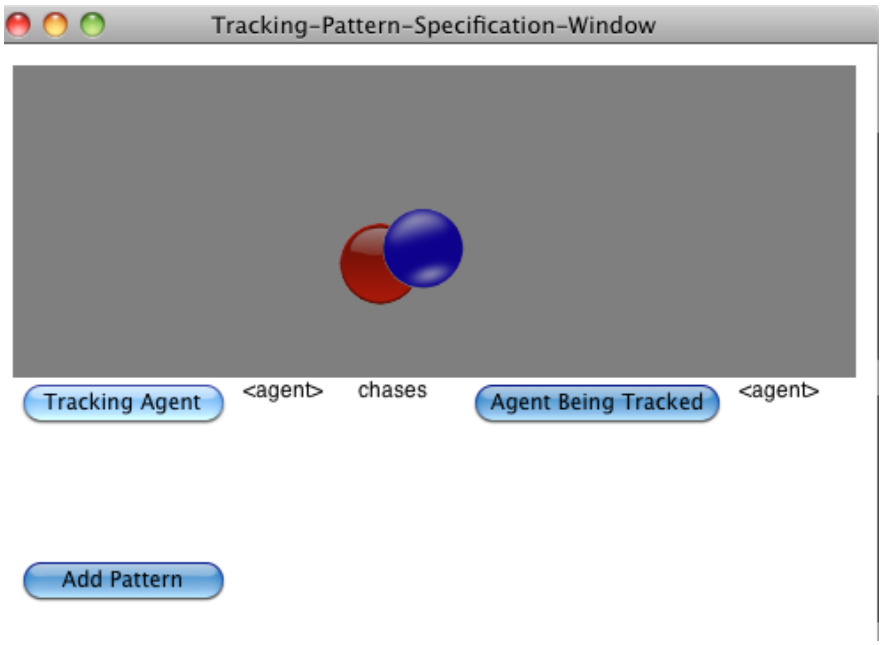


Figure 12: Simulation Creation Toolkit Tracking Pattern Palette

There are a few things to explain about this window. First of all, there are two balls in the top window. These balls act out a selected pattern; in this case, one ball chases or tracks the other ball. These generic agents acting out the pattern are called "interacticons" and will be covered more in-depth later in the thesis. Directly below the top window there are two buttons. The left button allows you to specify the tracking agent. The right button allows you to specify the agent being tracked. For our program we will select the Ball-python agent as the tracking agent and the Rat agent as the agent being tracked. The following figure depicts the palette after we have selected our agents.

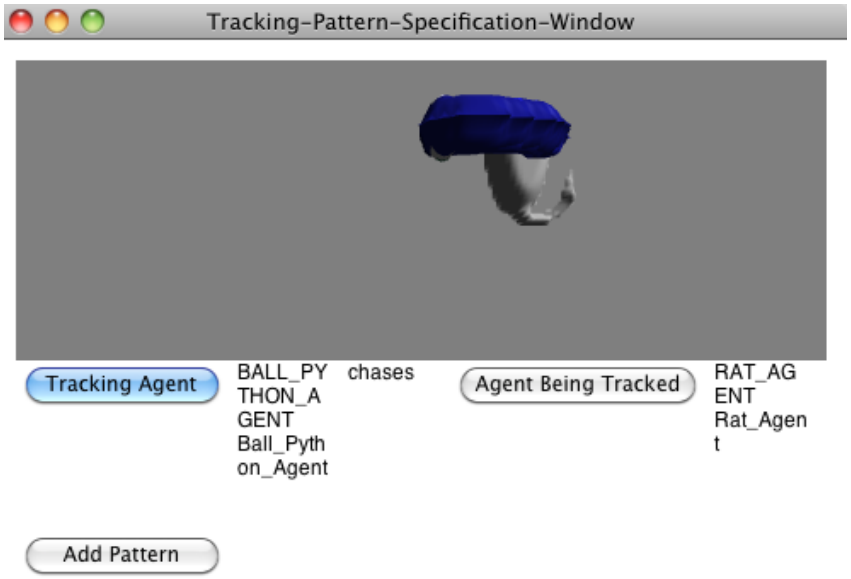


Figure 13: Simulation Creation Toolkit Tracking Palette With Ball Python And Rat Agent Selected As Tracking And Tracked Agent Respectively

Notice that in Figure 13 the top window wherein the two balls were enacting the tracking pattern are replaced by our agents acting out the pattern. Also notice that under the window we have a sentence that states the Ball_Python_Agent chases Rat_Agent (the reason the names are repeated in Figure 13 is because one is the name of the shape and one is the name of the agent. See section 1.4.1). Now that we have selected our tracking agent and our agent to be tracked we can hit the "Add Pattern" button in the bottom left of the window.

At this point we are given the following Pattern Specification Palette.

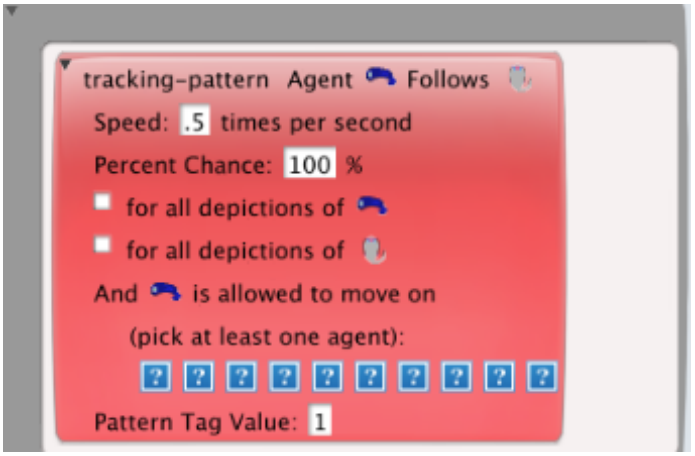


Figure 14: Pattern Specification Palette For Tracking Pattern

Figure 14 depicts the Pattern Specification Window that appears after we have selected a pattern. It gives us choices, starting from top to bottom, such as the tracking and the tracked agent (these can be changed at this stage), how fast we want the Ball-python to track the Rat agent ("Speed"), A chance that the Ball Python will not move towards the Rat agent this update ("Percent Chance)", if we want all the shapes of the Ball-python agent to track and all shapes of the Rat agent to be chased ("for all depictions"), and most importantly for our purposes, what background agent(s) the python is allowed to move on. We will select the Tile agent as our only background agent yielding the following picture.

Figure 15: Tracking Pattern Palette With Tile Background Agent Selected

At this point we are finished with implementing this pattern. As we were making all these selections in Figure 13, 14, and 15 the Simulation Creation Toolkit, in real time and behind the scenes, was adding the correct AgentCubes methods, rules, conditions, and actions to our three agent behaviors to make the Tracking pattern work correctly. What normally takes a day or two for students to create has been reduced to a less than five minute task given that they can navigate a menu of interacticons acting out various patterns. This introduces the power of this tool to quickly enable students to create complex agent interactions for their simulations.

The question still remains, is this something that students have shown a potential to understand? If students are shown an interacticon of two generic agents acting out a pattern, can a student really make the analogy between interactions they see and the interactions they want to implement? To put this another way, can students abstract out the agent interactions from the behaviors of the agents themselves? The next section describes a feasibility study that investigated this idea.

1.5 Simulation Creation Toolkit Feasibility Study

In order to show that the idea of students being able to map a given science phenomena that they see onto Computational Thinking Patterns acted out by two generic agents is not untenable, a small-scale feasibility study was run at Centenary Middle School. The feasibility study consisted of showing students video of six different Computational Thinking Patterns (see section 1.2) in both science simulations and game programming. Students were given a box of cardboard shapes (squares, circles, and triangles) and asked to act out the shown phenomena. As students played out the onscreen interaction with cardboard tokens, their hands were recorded. Eight students were run through the study, four sixth graders with little or no prior programming experience and four seventh graders who had completed multiple AgentSheets projects before.

It should be noted that in previous research participants in the iDREAMS Summer Institute at the University of Colorado Boulder, consisting of middle school teachers and community college students, were asked to identify patterns they had previously programmed in out of context video clips. Through this study it was shown that middle school teachers and community college students could readily identify the pattern from the video [26]. To put this another way, participants could identify the Computational Thinking Pattern they would need to implement the interaction in a video given that they had previously programmed the pattern. In contrast, this feasibility study investigated the user's ability to map the onscreen phenomena into icons by recreating a generic version of similar behaviors. Furthermore, the idea behind this feasibility study is that students could enact the pattern even if they had not necessarily implemented it before in

AgentSheets. One can think of the cardboard tokens students use to act out given interaction as parts of the Simulation Creation Toolkit animation.

The six interactions the eight students were asked to act out are as follows:

1. Transferrin (a glycoprotein that binds iron) transporting an iron molecule
2. Transferrin launching iron
3. A Cheetah tracking a gazelle
4. Amylase cutting starch
5. Truck being absorbed by a tunnel in Frogger (from AgentSheets game)
6. Agent pushing a box in Frogger (from AgentSheets game)

Figure 16 depicts the video of Transferrin transporting iron along with video screenshots of a 6th grade student acting it out.

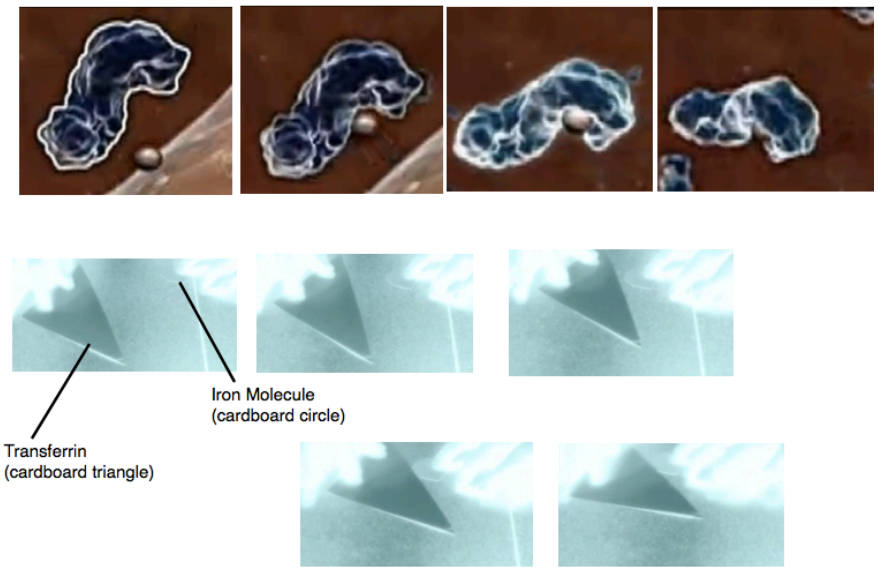


Figure 16: Screenshots of a Video clip of Transferrin transporting iron (above) with screenshots from a video clip of a sixth grade student trying to act out this interaction (below). For the most part, all students were able to act out all interactions.

Students, regardless of their background for the most part, were not only able to act out the patterns, but were consistent in how they acted out the phenomena. Some students acted out the patterns by using multiple cardboard tokens to roughly recreate the shape of the agents they saw. For example, some students approximated the shape of transferrin by combining the triangles, squares, and circles together. Similarly, some students actually tried to create the cheetah and gazelle in part 3 of the study. The sixth graders had never seen the amylase example before, many of their depictions of the starch chain involved circles together in a row. The seventh graders had completed the amylase unit earlier in the year (and therefore, had seen the example I showed them before). Many of their starch depictions were more complex using multiple tokens to create the complex links they had seen in class. When the tunnel absorbed the truck many of the sixth graders, who had never programmed in AgentSheets before, had the cardboard token representing the truck actually go inside the tunnel. The seventh graders, who had programmed in AgentSheets, tended to quickly remove the truck when it was in front of the tunnel. This might be because the seventh graders know that in the actual implementation of the truck/tunnel interactions in AgentSheets (also called the Absorb pattern), the truck gets deleted when it is in front of the tunnel and, therefore, tried to re-enact that deletion by quickly removing the truck agent.

Aesthetic differences between agents and actions in student re-enactments aside, students on the whole were able comprehend and act out the various interactions, whether it was transporting, launching, cutting, linking, pushing, or absorbing, in a similar manner. Based on these findings and the findings in previous research, one could surmise that it would not be infeasible for students to instead map interactions they are studying in their

current Life Science unit to a set of generic animated icons reenacting these patterns. Using the results of this feasibility study one can argue that using Computational Thinking Patterns to create simulations in a tool similar to the Simulation Creation Toolkit is an avenue of research that should be investigated.

1.6 Project Description

This project will investigate the benefits of having students program at the Computational Thinking Pattern level. This will be accomplished through the creation and testing of a tool called the Simulation Creation Toolkit (SCT) wherein students will create simulations by using combinations of Computational Thinking Patterns (see section 1.2) and with the Simulation Creation Toolkit implementing the patterns automatically in AgentCubes.

1.6.1. Research Questions

Presently, we have defined Computational Thinking Patterns as the units of transfer between games and science simulations. An implicit assumption in this definition is that it is useful for students, creating simulations pertaining to science phenomena, to think of the simulation as a combination of these high level patterns. This project will investigate this assumption with two main questions regarding the efficacy of Computational Thinking Patterns in the domain of science simulations by creating and testing a tool called the Simulation Creation Toolkit (SCT), wherein students program by primarily using high level Computational Thinking Patterns. The main research questions are as follows:

RQ1: Can students programming at the Computational Thinking Pattern level successfully create simulations of scientific phenomena they are presented within class?

RQ2: Does programming science simulations using high level Computational Thinking Patterns lead to better student conceptualization and understanding of the material they are being taught?

1.6.2 Remaining Sections

The remaining chapters will aim to answer these above research questions. The chapters that follow describe this specific area of research, the tool developed and the study run on the Simulation Creation Toolkit. Chapter 2 will focus on prior research in this area. Chapter 3 will describe the Simulation Creation Toolkit more in-depth as well as a discussion of its benefits and shortcomings. Chapter 4 will describe the experimental setup for the study. Chapter 5 will present the results of the study. Chapter 6 will discuss the results more in-depth, appraise the system and present any other pertinent discussions on this research. Chapter 7 will present possible future research and concluding statements.

CHAPTER 2

2. PREVIOUS RESEARCH

This section will briefly look at selected related research that covers high-level concepts touched upon in this thesis as well as similar projects and projects that incorporate similar concepts. Specifically, this chapter will look at the use of high level patterns and their use in end-user game design as well as end-user simulation creation. Finally, this chapter will outline why this thesis is a novel idea.

2.1 High Level Patterns As A Strategy Of Implementation

Using patterns as a strategy of implementation has a rich history both inside and outside of the computer science domains. The inspiration for much of this strategy comes from the book "A Pattern Language: Towns, Buildings, Construction" [31]. The idea behind "A Pattern Language" is to create a new high-level language, based on patterns, to enable people to create parts of towns, cities and urban centers. There are two interesting points that relate this book to the research in this thesis. "A Pattern Language" aims to use patterns to enable users, with little or no prior architecture and city planning experience, to effectively build the infrastructure of a given space. To this end, each pattern is an implementation tool that helps solve a given set of issues.

The second related aspect of this approach is that patterns start with general implementations but are fully implemented through sub-patterns or characteristics that each general pattern takes on. This relates to this thesis as the patterns in the Simulation Creation Toolkit can have multiple implementations that each do something slightly different but are all under

the category of a specific pattern (see section 1.4.2). One significant difference between the ideas in this text and the Simulation Creation Toolkit is that the patterns to build towns, for obvious reasons, are not automatically implemented upon pattern specification. The user plans a given space at a pattern level but then must implement these patterns themselves using the low level tools of architecture and construction.

The seminal text "Design Patterns: Elements of Reusable Object-Oriented Software" takes this idea of breaking down problem solutions into various design patterns and applies it to the domain of Computer Science and specifically, object-oriented (O.O.) software creation [32]. Similar to "A Pattern Language", each pattern starts with a set of problems/constraints that must be met and then presents a pattern that could be used, at a high level, to fulfill these constraints as stated in the following quote that relates Object Oriented design to "A Pattern Language".

> "Even though Alexander was talking about patterns in buildings and towns, what he says is true about objet-oriented design patterns. Our solutions are expressed in terms of objects and interfaces instead of walls and doors, but at the core of both kinds of patterns is a solution to a problem in context [32]."

The reason that this strategy of patterns is interesting in the context of object-oriented design is partly due to the idea that "expert designers" come across solutions over years of experience involving trial and error. Encapsulating high-level patterns that would not be readily apparent to a novice user and yet solve common problems in a given domain is exactly the aim of Simulation Creation Toolkit. The authors of "Design Patterns" go onto state the following.

> "Design patterns that describes simple and elegant solutions to specific problems in object-oriented software design. Design

> patterns capture solutions that have developed and evolved overtime. Hence they aren't the designs people tend to generate initially. They reflect untold redesign and recoding as developers have struggled for greater reuse and flexibility in their software. Design patterns capture these solutions in a succinct and easily applied form [32]."

This quote brings up two interesting ideas. The first is the idea that a user might tend to not initially come up with a certain pattern implementation as a solution to a set of problems/constraints. This relates to many of the Simulation Creation Toolkit patterns; for example, a user might want one agent to track another agent but would probably be hard-pressed to figure out and apply the ideas of diffusion and hill climbing in agent behaviors without prior experience (see sections 1.4.3, 1.4.4).

The second idea is the ability for the pattern to be flexible. In the Simulation Creation Toolkit this creates a tradeoff relating to what elements of a pattern a user should be able to specify while still keeping the pattern simple enough to use and understand. To put this another way, how can we present the user with as many useful avenues as possible to create the Generate pattern (see section 1.4.2), for example, for their specific purposes, without having to present the user with an enormous and possibly overwhelming amount of alternatives within a given pattern? This tradeoff is a byproduct of the Simulation Creation Toolkit implementing the pattern directly from a user's high level choices rather than having the user implement the pattern themself to their specification (as is done in "A Pattern Language" and "Design Patterns"). The implications of this will be talked about more in the next chapter.

2.2 Human Perception Of High Level Patterns

The genesis for the idea of generic actors enacting specific patterns, described as "interacticons" for the purposes of the Simulation Creation Toolkit (see section 1.4.4 and Figure 12), owe much to experiments developed by Albert Michotte in his book "The Perception of Causality" [33]. It should be noted that the specific interacticons used in each pattern for the Simulation Creation Toolkit will be covered more in-depth in Chapter 3. Michotte did numerous experiments on how humans perceived a given interaction. To this end, Michotte created mechanical devices that enabled him to alter movements of given objects on a projection screen. He would then ask subjects what they perceived to see while slightly changing variables such as the timing of object movements relative to each other, the size of the objects in the interaction, the plane the objects exist in, and the speed of the objects themselves among other things. From this he could record the moments when the subject perceived a certain interaction between objects and when a given interaction impression was lost. Specifically, Michotte states in his introduction

> We need to know that things can be moved, e.g. by pushing them, causing them to slide, lifting them, or turning them over, by hurling, breaking, bending or folding them, by leaning on them, and so on. . .Although these events all have a spatial and kinematic aspect, the most important feature about them is that they imply *functional relations* between objects [33]."

The intimate relationship between Michotte's experiments and interacticons exists because the success or failure of generic agents exhibiting a pattern depends on the ability of the user to abstract out agents in a given interaction while preserving the interaction itself. To put this another way, the user must perceive these generic interactions as a pattern being acted out

in order for the Simulation Creation Toolkit to be usable. This idea was partly investigated in the Feasibility Study described above (see section 1.5).

Michotte is not concerned with the implementation of patterns, however, many of the pattern-phenomena the interacticon palette in the Simulation Creation Toolkit attempts to recreate, via generic agents enacting interactions, are analogous to phenomena Michotte uncovers in his various experiments. For example, Michotte talks about the "Transporting Effect" on page 151:

> "What is involved, as we know, is a separation of the systems of reference; the transported object is referred to its vehicle, and the vehicle is referred to the surrounding field. To obtain the Transport Effect, therefore, it is necessary to ensure that the transporting object isolates the transported object from the space around it. An example of this occurs when one object is inside another, or when there is an obvious break between the object and the surrounding space, as in the case of an apple handed to someone on a plate [33]."

Similarly, the interacticon for the Transport pattern in the Simulation Creation Toolkit depicts a small circle jumping on top of a bigger circle and then moving with the same velocity as the bigger circle (much like the apple on a plate analogy used by Michotte, see section 1.2).

Another example is experiments 74 and 75 outlined on page 232 [33]. In the first instance (experiment 74) two round objects come together and one of them disappears. In the second instance (experiment 75), two round objects come together and one of them changes color. These two experiments are very similar to the interacticons for the Change pattern and the Absorb pattern (see section 1.2). There are many more examples of interacticon animations of patterns mirroring and exploiting perceptions discovered in various Michotte experiments; the big picture of how this relates to this project relies

on the idea of users being able to perceive the patterns from these generic agent interactions.

2.3 Use Of High-Level Patterns In End-User Game Design

High-level patterns have often been linked to game design as a way to make the implementations of given interactions shared among a variety of games concrete. As noted in previous sections, the emergence of Computational Thinking Patterns started out as patterns that students first learn to implement in game programming and then, later, transferred to the creation of science simulations (see section 1.2). Therefore the use of patterns in game design is similar and often overlaps with the use of patterns in science simulations.

One big proponent for the use of game design patterns is Steffan Bjork who wrote the book "Patterns In Game Design" [34]. This book specifically aims to classify many patterns that are commonly used in all aspects of game design; therefore, much of it is outside the scope of this thesis. However, one section of the patterns presented relates closely with the agent interactions outlined in Computational Thinking Patterns. In the introduction of the book, the following is stated.

> For the sake of this discussion, we define gameplay simply as the structures of player interaction with the game system and with the other players in the game. Thus, gameplay includes the possibilities, results, and the reasons for the players to interact within the game. For example, the gameplay of Space Invaders consists of the player moving the ship left and right at the bottom of the screen, trying to shoot wave after wave of invaders and at the same time avoiding their incoming fire [34]."

The book, in its Pattern Collection section, outlines patterns that one can see easily relate to the interactions outlined in the above Space Invaders quote.

For example a user might use a Movement pattern for both the ship and alien characters; furthermore, a user might use the Shoot and Kill patterns to have the ship eliminate enemy agents in Space Invaders. Many of the interactions we would find in the Simulation Creation Toolkit can be found in the "Actions and Events Patterns" section of the book which outlines patterns such as "Spawning" and "Movement" wherein Spawning would be akin to the Generation Computational Thinking Pattern and Movement would relate to a variety of movement patterns such as the Tracking or Random Movement pattern. Similar to the books outlined in section 2.1, this book enables users to plan games and gives them ideas for implementation.

One idea that is put forth in this book, and has major ramifications in the implementation of the Simulation Creation Toolkit, is that combining patterns could change each pattern present in the combination. "Patterns in Game Design" specifically introduces the idea of "modulating patterns" wherein the presence of a first pattern, when a second pattern is implemented, yields a change in the first pattern to accommodate this second pattern. When describing modulating patterns, the book states the following.

> "Modulates: The first pattern affects aspects of the second pattern in a way that influences gameplay. The modulating pattern works within a limited design space that is bounded by other restrictions regarding gameplay. . . the modulating pattern aids in fine tuning another pattern.[34]"

The idea of modulating patterns and how they relate to the Simulation Creation Toolkit will be made clearer next chapter; for now one can think of a simple example wherein a Agent A tracks Agent B and Agent A also transports Agent C. In this case the Transport pattern modulates the movement altering the Tracking pattern of Agent A; namely, it is not enough

for Agent A to simply move towards Agent B anymore, but rather, Agent A must also check if it is under Agent C, and, if it is, must transport Agent C with it as it moves towards Agent B. Thus, this idea of modulation plays a huge part in how certain patterns interact with each other.

EEClone takes this idea of creating games through patterns and uses it as a strategy to enable beginning programming students learn the implementation of various design patterns in object oriented programming [35]. Specifically, EEClone enables students to create a facsimile of a video game called 'Every Extend', and ,in doing so, the students implement six O.O. patterns. The patterns that are implemented in EEClone have analogues in Computational Thinking Patterns. For example, in EEClone students implement "The State Pattern" wherein a character changes from one state into another state (i.e. Pacman changing when a power pellet is eaten). This is very similar to the Change Computational Thinking Pattern. Similarly "The Controls Pattern", wherein a user can add event based handlers to their game loop, is similar to the Keyboard Control Computational Thinking Pattern. The aim of EEClone is distinctly different from the goal of the Simulation Creation Toolkit in that EEClone is trying to get students to implement certain O.O. patterns that happen to lend themselves to a given game context. However, given this idea, it is no surprise that there is overlap between Computational Thinking Patterns, which started in the context of game creation, and EEClone.

2.3.1 High Level Patterns For Domain Orientation In End-User Game Design

Often patterns are used to domain-orient a given system towards a given user goal. We will now review a few examples of these systems briefly.

One early example of Domain Orientation towards system design involves the idea of Construction Kits [36,80]. Researchers Fischer and Lemke set forth the following argument.

> "To provide the user with the appropriate level of control and a better understanding, we have to replace human-computer communication with human problem-domain communication, which allows users to concentrate on the problems of their domain and to ignore the fact that they are using a computer tool [36]."

The idea behind Construction Kits is to supply the user with high-level building blocks that, by combining, the user can create a variety of different projects in a given domain much more easily than would otherwise have been possible. For example, the researchers talk about a software product called the Pinball Construction Kit for the Apple Computer. The Pinball Construction Kit provides users with flippers, bumpers, bouncers, and all the other items one might find in a pinball machine such that the user could create their own pinball game. By using these building blocks the researchers argue that traditional problems with human computer interaction are alleviated. These problems include helping to break the complexity barrier, utility barrier (ratio of value to energy expended), automatically taking care of things the user does not want to do, and mirroring the abstractions of the application domain [36]. In this sense, the aims of the Simulation Creation Toolkit and Construction Kits in a particular domain are very similar. Furthermore, one could interpret Computational Thinking Patterns as the "building blocks" of the Simulation Creation Toolkit much like the paddles and flippers etc. are the building blocks of the Pinball Construction Kit. Therefore, though the Simulation Creation Toolkit is meant

for a domain much bigger than traditional Construction Kits, one can see how interrelated the ideas are.

Another example, closely related to the Simulation Creation Toolkit, occurred in AgentSheets. In earlier iterations of AgentSheets, researchers Iannidou and Repenning used the idea of domain orientation with agent behaviors was to help students create simulations [37]. The aim of the study done in [37] was to have students create simulations pertaining to animals in a given ecosystem, and later, protesters in the Montgomery, Alabama Bus Boycott. The researchers in this project found that getting students to create the behaviors from scratch, for both of these simulations, was too time consuming for first-time users and students. Therefore, the researchers decided to build in "Domain-Oriented Commands" and "Domain Oriented Templates." In the ecosystems project, called "EcoWorlds", the researchers stated the following regarding "Domain Oriented Commands".

> "In EcoWorlds, defining the predator-prey interactions is a major component of the programming; to support this activity, we provided a number of domain-oriented commands. For example, rules that enable a predator to eat are stated as, "If I can select food <description of prey, based on features> then try to eat it, because I am <description of self, specifying why I can eat this prey>." This set of commands replaces more basic actions, such as "see" and "erase," with the specific actions of selecting food and trying to eat it [37]."

To put this another way, in an effort to save time and focus what students were creating, the researchers found it helpful to make specific rules that that readily lent themselves to the creation of a given simulation type. Instead of users creating the predator eating the prey from their individual conditions and actions, the user could select the whole rule, which was phrased in the context of the simulation, and then specify the predator and

the prey agent. Similarly, "Domain-Oriented Templates" are agents preloaded with methods, rules, conditions and actions wherein users fill in what agents are involved in the specific rules.

One could think of the ideas behind Domain-Oriented Commands and Domain-Oriented Templates as the precursor to programming at the Computational Thinking Pattern level. The agent behaviors and the rules are very specific to a given simulation, but it is a set of higher-level rules that enables quicker user creation of a simulation. Furthermore, the motivations for creating this high level rule are very similar to the motivations for this thesis outlined in section 1.3.

The main difference between this and programming at the Computational Thinking Pattern level is that Computational Thinking Patterns are more generally applicable. The same Computational Thinking Patterns can be combined across a variety of simulations. This is partly due to the way Computational Thinking Patterns themselves emerged; as explained above, they were patterns that students learned in game programming and could transfer to simulations (see section 1.2). Another difference is that Computational Thinking Patterns can be used in combination with one another to create two different versions of the same simulation. Furthermore, the specifications of patterns implemented through the Computational Thinking Pattern level have more choices such as including a percent chance, having multiple depictions or just one depiction enact the pattern, adding multiple blocking agents to movement etc. Thus, students are specifying much more than just what agent happens to be in a Domain-Oriented Templates. Despite these differences the two ideas are both strategies towards solving a similar problem in the classroom.

Similar to the above approach, Storytelling Alice is another example of domain orienting to facilitate creation [38, 66]. Storytelling Alice takes the Alice system of end-user game and animation creation and adds elements of functionality that enable users to more easily create narratives. To this end, Storytelling Alice enables users to do things like change scenes, animate characters, add emotional expressions to characters, enable characters to talk and think (i.e. add word and thought bubbles), and enabling character locomotion in Alice more simply. Storytelling Alice shares the idea of domain orienting an existing end-user software tool dedicated for game creation to a specific purpose i.e. creating stories. Similarly, the Simulation Creation Toolkit can be thought of domain orienting AgentCubes to enable easier creation of simulations. Much of the ideas expressed in [38] are shared; things like students would not reasonably discover how to implement a given interaction and thus a higher level approach may enable easier creation within a given domain

2.4 Simulation Tools For Education

In addition to high-level patterns being used for game design, there are many approaches to using agent based and non agent based tools to integrate simulations in the classroom. Some examples include kidsim and swarm among others [64,65,70]. This topic is too big to cover fully in this thesis but what follows are three different illustrative strategies for incorporating simulations into the classroom.

One strategy to integrating simulations in the classroom is by using individual pre-made simulations that students can explore for a given topic. The PhET project at the University of Colorado Boulder, directed by Nobel

laureate Carl Weiman, is an example of this strategy [39,40]. The PhET project has an increasingly growing suite of simulations in Physics, Biology, and Chemistry. For example, a PhET simulation might explore the ideas about different characteristics of waves such as water waves, light waves and sound waves [40].

In this strategy students are often given "driving questions" that guide their exploration of the simulation. Students can be given varying levels of freedom to explore this simulation ranging from completely open to completely guided levels of direction. In previous research it has been shown that PhET simulations are most effective if students are able to make conceptual analogies with the subject they are exploring and things they have previously encountered [40].

There are many differences between the strategies employed by PhET and the idea behind the Simulation Creation Toolkit. The main difference is that students do no simulation creation; therefore, in terms of thinking computationally, the creator of the PhET simulation, rather than the users of the PhET simulation, are the ones dealing with problem formulation, representing phenomena through abstractions, dealing with algorithmic thinking, and generalizing the problem to other domains wherein a solution can be found (see section 1). The Simulation Creation Toolkit, in a sense, is more aggressive as it aims to have students create the simulation themselves; however, it is also more unfocused because the simulation students create is not necessarily gamed to have a student answer a specific question about a given phenomena. Also, as will be pointed out later, if students create an incorrect simulation using the Simulation Creation Toolkit and do not realize this, they might actually garner an incorrect representation of the world. PhET simulations on the other hand are

ostensibly made by "experts" rather than the users themselves, and so, one would hope that the PhET simulations created are more or less correct in terms of the driving questions the PhET creators want the students to answer. However, this strategy eliminates students figuring out what parameters in a given system are important to include for simulation modeling and manipulation in favor of providing students with the system parameters explicitly.

One similarity between PhET simulations and the Simulation Creation Toolkit is that both systems are most effective when students can make analogies between the phenomena they are studying with something they have already been exposed to [40]. In the Simulation Creation Toolkit, however, the analogy is a requirement for creating the simulation itself. Therefore, at the point the students start experimenting on their given simulation, they have already attempted to explicitly make this analogy. In contrast, a given PhET simulation hopes that the simulation is presented in a context with enough scaffolding that enables the student to easily make any analogies that are necessary for them to better understand the concepts behind the driving questions.

A closer strategy to the Simulation Creation Toolkit involves students not only working on pre-made simulations, but actually, participating in the simulation creation process. The Starlogo "Adventures in Modeling" challenge as well as research on Starlogo TNG provides us with examples of each [41,42]. Starlogo is an agent-based modeling tool, and, much like AgentCubes, enables users to add behaviors to various agents. Akin to AgentCubes drag on drop behaviors, Starlogo TNG, another Starlogo system, aims to have users use graphical drag and drop rules to give agents behaviors [42]. The Adventures in Modeling challenge used 5th grade and 7th grade

students to manipulate pre-made systems to further understand a given topic and then design their own open-ended systems that exhibits a given interaction. For example, students were asked to modify a pre-created epidemiology simulation wherein agents get sick and die with different percentages much like the epidemiology simulation outlined in section 1.2. Later on in the study, students were asked to create something open-ended like "designing a project where creatures interact with their environment" [41]. In this study the authors found that student ideas of what could and could not be represented as a model were broadened. One could argue that the pre-made simulation strategy falls into the same pitfalls as the PhET simulations, and in fact, researchers working on Starlogo TNG tend to agree as is illustrated by the following quote.

> "While pre-built simulations can provide students with accessible visualizations, immersive learning environments, and the opportunity to analyze data from virtual experiments, they cannot provide them with the freedom to express and explore their own understanding of a given phenomenon through simulation use, nor can they provide much insight into the use of simulations in the scientific enterprise generally. In order to do that, students must be able to experience the full spectrum of interactions with simulations which combine science and engineering, including: studying and analyzing existing simulations, understanding and redesigning simulations, and even building simulations themselves [42]."

In Adventures in Modeling, when students did in-fact attempt to design their own open ended projects it was met with difficulties [41]. For example, the researchers decided to give students a task rather than keeping the problem statement open-ended. Students were given scaffolding such as starter projects and a more defined direction (i.e. such as "design a traffic

simulation") in order to get them started. The results of the open-ended design challenges are summarized by the following statement.

> "First, it shows that many of the principles of complex systems are not too complicated to be integrated into the classroom, even at the elementary school level. Young students can start to develop an understanding of some of the principles of complex systems, and the skills required to understand them . . . Based on the study results, continued application of the Adventures in Modeling curriculum at either of these grade levels would likely be enhanced by a greater repertoire of scaffolding tools that would allow students to select from a variety of starting points [41]."

This quote directly motivates the Simulation Creation Toolkit as it states that simulations can be effectively integrated into the classroom given that students are provided with the correct scaffolding, including tools, that make the act of simulation creation attainable. The strategy employed by the Simulation Creation Toolkit is the use of high-level Computational Thinking Patterns to help lower the barrier hopefully enabling simulation creation among students.

As mentioned above, Starlogo TNG attempts to have students create simulations by making it easier for students to add behaviors to agents. Namely, Starlogo TNG enables students to use graphical drag and drop rules, called "programming blocks" with conditions and actions, to make rules [42]. Using Starlogo TNG, however, runs into the same problems outlined in section 1.2 again, namely, it is still not at a high enough level to have students program a great portion of the simulation from scratch. Therefore, in [42], for example, students are asked to update parts of a given simulation with behaviors wherein they model things like the spread of forest fires, but they are adding low level behaviors within a limited scope of what has

already been created. The Simulation Creation Toolkit takes the ideas promoted in a tool like Starlogo TNG and attempts to expand them in order to encompass all the benefits of creating simulations, modifying simulations, and running experiments on simulations.

A final example comes from Randall Davis at M.I.T. wherein users use free-form informal sketches within a defined context using a tool called "Magic Paper" to make sketches that act as simulations [43]. For example, using Magic Paper a user might quickly sketch out a given circuit that can be read into a Spice (a circuit simulation program) file and have experiments run on it. A person might also make a sketch of a ramp with an object resting on it, and by defining the forces at work, run a simulation wherein the object animates by rolling or sliding down the ramp.

Much of this research is centered around sketch interpretation, namely, what is the correct interpretation of a given sketch regardless of how it is drawn. However, the idea behind quickly creating prototypes of simulations is related to the idea behind the Simulation Creation Toolkit. Magic Paper aims to achieve this by directly interpreting user sketches wherein the Simulation Creation Toolkit leaves the interpretation of interacticons up to the users themselves. For a simulation in Magic Paper to work correctly the program must understand what is being drawn by a user; in the Simulation Creation Toolkit the onus is on the user to create the analogy between phenomena they want to simulate and the agent interactions that will correctly create this simulation. In this sense one can think of the Simulation Creation Toolkit strategy as somewhere in-between Magic Paper wherein ideally one would be able to draw anything to be interpreted by the system and the AgentCubes/Starlogo TNG strategy of lower level behavior creation in an agent-based system.

2.5 Other Related Research

In addition to the above research, there is other research that is less related to the Simulation Creation Toolkit but should be reviewed so the reader has a better idea of the general areas the Simulation Creation Toolkit resides in.

Additional end user game creation tools, that will be touched on briefly, include Squeak, Scratch, Greenfoot, and the aforementioned Kodu (see section 1.2). It should be noted that this is by no means an exhaustive list and additional end-user tools for education can be found at [5,44]. Squeak enables students to recreate interactions they see by using higher level commands built on a smalltalk instruction set; Squeak has been used in educational settings, such as 6th grade classrooms, to visualize problems and derive solutions [45]. Scratch enables users to create games using graphical drag and drop rules called "building blocks" [46]. These building blocks fit together like puzzle pieces and dictate the behavior of onscreen characters. Much of the research into Scratch involves the ability for users to share games online and "remix" already created games [47, 48, 72]. Greenfoot, on the other hand, partners the onscreen object approach with Java as the mechanism for adding behaviors to objects as opposed to graphical drag and drop rules [49]. Greenfoot has also been used as a simulation creation tool, for example, as described in [50]. Kodu is a game creation tool from Microsoft Research. Kodu consists of many premade characters that users can add to their games as well as behaviors for these characters based on physical senses such as vision and hearing [51,62]. Kodu games have the ability to be ported to the XBOX gaming console. All of these tools share characteristics with the Simulation Creation Toolkit and AgentCubes in that they all depend on adding behaviors to agents or objects in a world. Furthermore, Kodu, for

example, has some overlap with the Simulation Creation Toolkit in that the behaviors of in-game characters are at a higher more intuitive level (see section 1.3.2).

2.6 What Makes The Simulation Creation Toolkit Different

The Simulation Creation Toolkit is an investigation into the efficacy of Computational Thinking Patterns as the building blocks for simulation creation. Therefore, though there are simulation creation tools in existence, and game programming tools that use various methods of domain orienting or using patterns as high level programming blocks, none of the above systems use patterns quite like Computational Thinking Patterns. Computational Thinking Patterns have previously found to be useful in transferring knowledge of game design into the creation of simulations [27,10]. Computational Thinking Patterns are already being used to appraise what students have achieved in previously created games, and the teaching of certain Computational Thinking Patterns has been identified as a "watershed moment" in classrooms wherein students' subsequent programs change drastically after they learn a given pattern [52,53,63].

Computational Thinking Patterns give users the freedom in their program creation they might not initially have given time constraints and limited programming expertise. One wonders if Computational Thinking Patterns can be used directly as the building blocks of simulations given that students can discern the scientific phenomena they happen to be studying and relate it to the interacticon palette. The remainder of this paper will not only outline the Simulation Creation Toolkit in-depth, but also, serve as a

first step towards answering this question through a study conducted on the Simulation Creation Toolkit.

CHAPTER 3

3. THE SIMULATION CREATION TOOLKIT

This chapter will describe the Simulation Creation Toolkit. First we will give an overview of the tool's design goals. Next we will introduce the patterns included in the toolkit.

The Simulation Creation Toolkit is extremely complicated to describe. This is due to the fact that every part of the system is interdependent, meaning it is difficult to talk about user selections in the interface without also discussing what these selections mean in terms of the AgentCubes code created. Furthermore, pattern implementations themselves can become extremely complicated as a user makes selections. Therefore, instead of discussing each part of the system separately, three illustrative top-down examples will be presented in order to better show how the different system elements work in concert. Next, we will outline every pattern included in the toolkit reviewing their specification choices as well as discuss each pattern's implementation implications. After that, we will briefly present a few examples of simulations that can be accomplished using this tool and a few examples of simulations that might be difficult to create. Finally, we will discuss the user interface elements including the system windows and interacticons.

All the code for the Simulation Creation Toolkit is written on top of the AgentCubes code-base using XMLisp, a modified version of the Lisp language[6] [77].

[6] http://code.google.com/p/xmlisp/

3.1 The Big Picture Behind The Simulation Creation Toolkit

The idea behind the Simulation Creation Toolkit is to allow students to create simulations by using Computational Thinking Patterns to add behaviors to agents as opposed to implementing patterns using lower-level if/then conditionality rules (see section 1.4). To this end, users select the interactions they want between agents rather than what each agent should do individually to create these interactions. Looked at another way, the summative action, enacted by one or many agents and often consisting of many behavior rules, is the important mechanism for adding behaviors.

Using the Simulation Creation Toolkit, users are provided with a palette of "interacticons" wherein generic animated agents act out a particular pattern. Using this palette, users select patterns that dictate how agents will act during a simulation run. Computational Thinking Patterns are extremely general and can be implemented a variety of ways. One major challenge is to enable users the ability for implementing patterns as many ways as possible.

For example, let us say that we have one agent generating another agent (i.e. a gun agent creating bullet agents or a tunnel agent creating truck agents). It makes sense that we would use the Generate Pattern to accomplish this (see section 1.2). However, what is not clear is how the generation of agents will happen. Will creation happen once every so often? Does creation happen if the user hits a key? Is there a percent chance associated with this creation? What direction is this agent created in, or is it all directions? Do all the shapes of the creating agent accomplish this generation or is it just a specific shape that enacts this pattern (note that as mentioned above in section 1.4.1, the terms "shapes" and "depictions" can be used interchangeably for our purposes)? Do we want to keep track of every

agent-creation in a variable? Very quickly the implementation of the Generate Pattern gets more complex as one realizes the almost intractable number of choices and possible condition/action combinations associated with this pattern.

The Simulation Creation Toolkit diverges from classic pattern strategies in that it attempts to implement the patterns based on user selections rather than having the user implement the pattern her or himself (see sections 2.1, 2.3). It would be impossible to have every single choice for a particular pattern given to the user; thus, the challenge is to provide the user with as many choices as necessary to cover the greatest variety of and the most common pattern implementations. Ideally, this would be accomplished without having the pattern specifications themselves become overly complex. Furthermore, the implementation of patterns must be such that it can accommodate any prior or subsequently implemented pattern, including modulating patterns (see section 2.3).

There are four general guidelines that the Simulation Creation Toolkit strives to meet.

1)	Patterns should be easy to create, modify, and delete from a program.
2)	Patterns should allow for the most number of specifications to meet the needs of users.
3)	Pattern implementations should not infringe on prior or subsequent pattern implementations.
4)	Patterns implementations should work correctly with any modulating patterns.

Table 3: General Design Guidelines For Simulation Creation Toolkit Pattern Implementations

These guidelines were in part determined through meetings with my advisor Dr. Alexander Repenning. The first guideline is based on the idea that a user should not only be able to add and delete patterns, but, just as importantly, should not have to delete and re-implement a pattern if a mistake happens to be made. The second guideline refers to the idea that pattern specification should enable a wide variety of pattern implementations. As will be shown, when users are implementing patterns, code is being constantly and automatically added to AgentCubes; the third guideline states that this code should not invalidate a pattern the user previously implemented. Similarly, the fourth guideline states that modulating patterns should correctly alter prior and subsequent pattern implementations.

One important idea behind these guidelines is that a program created using the Simulation Creation Toolkit should always do something "sensible"; the program should always run in such a way that reflects a correct interpretation of the implemented patterns (there may be multiple correct interpretations). As will be discussed later when we look at each pattern in-depth as well as their implementation, this is not always feasible; therefore, it is the aim of the Simulation Creation Toolkit to get as close to this ideal as possible.

3.2 Patterns Included In The Simulation Creation Toolkit

Ideally, the Simulation Creation Toolkit would have every possible pattern as a choice for the user. Realistically, of course, this is not possible. The Simulation Creation Toolkit is an initial investigation into the efficacy of using Computational Thinking Patterns as a means of simulation creation. Therefore, the Simulation Creation Toolkit needs to provide users with

enough patterns to create a variety of agent-based simulations. Future research into an authoring tool enabling user pattern creation could be one solution to the issue of a limited pattern palette. Furthermore, as stated in section 1.2, Computational Thinking Patterns are a work in progress. With new patterns being discovered as more teachers use AgentCubes in their classrooms, and as the projects they undertake get increasingly diverse, a pattern-authoring tool might be a great way to accommodate the evolution of Computational Thinking Patterns.

Using meetings with both my advisor, Dr. Alexander Repenning, and Marco Cornacine (7th grade Life Science teacher, Centenary Middle School. see section 1.3) as guidance, a palette of patterns to include in the Simulation Creation Toolkit was determined. Additionally, the Simulation Creation Toolkit pattern palette is heavily influenced by the list of already discovered patterns outlined in section 1.2 as well as the Guaranteed Viable Curriculum at Centenary Middle School. This study is in large part an analysis of how useful programming at the pattern level can be. Thus, it makes sense that included patterns would consist of ones that were previously found to be effective. The following is a list of patterns included in the Simulation Creation Toolkit with a brief description of what they accomplish.

1) **Change**: One Agent/Shape Changes Another Agent/Shape Into A Third Agent/Shape.

2) **Absorb**: One Agent/Shape Makes Another Agent/Shape Disappear.

3) **Transport**: One Agent/Shape Carries Another Agent/Shape.

4) **Push**: One Agent/Shape Pushes Another Agent/Shape.

5) **Random Movement**: Enables An Agent/Shape To Move Randomly.

6) **Tracking**: One Agent/Shape Chases Another Agent/Shape.

7) **Keyboard Control Movement**: Agent/Shape's Movement Is Dictated By The User Hitting Keyboard Keys.

8) **Directional Movement**: The Agent/Shape Moves In A Given Direction Once Every So Often.

9) **Generate**: One Agent/Shape Creates Another Agent/Shape.

10) **Data**: Enables The Counting Of An Agent/Shape; i.e. Users Can Count The Number Of A Given Agent/Shape Instance Present In The World At A Given Time.

Table 4: List Of Patterns, With A Brief Description, Included In The Simulation Creation Toolkit.

The first four patterns are generally categorized as "collision" patterns, meaning they are triggered when two agents come together or are next to each other. The next four patterns are generally categorized as "movement" patterns, meaning they add motion to a given agent. The last two patterns are their own category. Generate might occur as a collision, for example, when a Fox Agent sees another Fox Agent in a given direction, it

might generate a new Fox Agent to simulate mating; however, Generate can also occur without a collision, for example, when something is generated on keyboard hit or when a stream of agents is generated once every so often. The Data pattern keeps a running tally of a given agent's instances present in the world at a particular time; thus, it is neither a collision nor movement pattern.

One point of confusion in this categorization is the inclusion of the Transport and Push Patterns as collisions. At first glance, both of these patterns seem more related to movement than collision. However, taking a closer look at these patterns, and the guidelines for collision categorization provided above, we see that both patterns involve two agents coming together before any movement can occur. One might correctly point out that the agent being Transported and the Agent being Pushed in these patterns are given movement. However, this movement is subject to the movement patterns already implemented in the Pusher Agent and Transporting Agent.

Push and Transport are examples of modulating patterns wherein, the nature of the Push and Transport depend on the movement pattern applied to the Pushing and/or Transporting agent (see section 2.3). In the Simulation Creation Toolkit, the implementation of the Transport Pattern, for example, does not actually give an agent locomotion. It merely states that regardless of the rules that dictate this agent's movement, if it happens to be under the specified Transported Agent, then it will transport that agent as it moves. Similarly, if Agent A Pushes Agent B, it means that if Agent A happens to move onto a square where Agent B currently resides, Agent B should move one space over in the direction Agent A is moving, and Agent A should move into the space where Agent B was previously located. Therefore,

for the purposes of the Simulation Creation Toolkit, these patterns affect the agent movement but do not provide the agent with the capability to move.

A central component of any pattern is how the pattern is specified. A pattern specification not only includes the agents involved in the pattern, but also, could include things like how often the pattern is executed, percent chance associated with the pattern, and the positions of agents necessary to trigger the pattern among many others. The specification of each pattern dictates how an agent will act under the direction of this pattern. In terms of simulations, specifications can be thought of as simulation or system parameters.

The same pattern can have many different implementations, which, in turn, lead to different agent behaviors. Specification choices limit what the user can and cannot implement in terms of a pattern. Therefore, in addition to choices of patterns to include, as important, is the decisions made as to what can be specified in a pattern. The following figure depicts the simplified workflow using the Simulation Creation Toolkit.

Figure 17: Simplified Simulation Creation Toolkit Workflow

Figure 17 can turn into a loop as users re-specify previously implemented patterns.

The description of the Simulation Creation Toolkit becomes extremely complex very quickly. Therefore, in order to get a clear basis of how the system works, the next few sections take a step by step top-down approach starting at the user selection level and ending at the corresponding

low-level AgentCubes code created. We will first present an illustrative example of adding a Random Movement Pattern to a simulation, going through the workflow depicted in Figure 17. As we add this pattern, we will talk in-depth about the user interface and AgentCubes' implementation elements. Every window and interacticon that make up the Simulation Creation Toolkit, with a description, can be found in Appendix B. After that, we will present an example of two modulating patterns on top of this simulation: Transport and Push, so the reader gets a better idea of how patterns interact with each other in the Simulation Creation Toolkit. For the remainder of this chapter, to avoid confusion, references to AgentCubes conditions and actions will be in bold face (i.e. "**see condition"**, "**move action"** etc.). Appendix A provides a reference for every AgentCubes condition and action along with a short description.

3.3 Illustrative Top-Down Example Of Simulation Creation Toolkit: Adding Random Movement To A Simulation

Let us say we have an AgentCubes Simulation consisting of 4 Agents: A Rat Agent with a Male and Female Shape, A Pizza Agent, A Box Agent, and a Background Agent. The following picture shows these four agents.

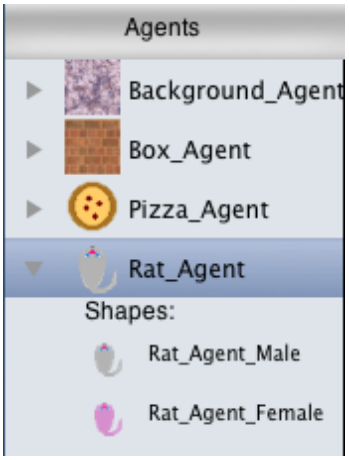


Figure 18: Simulation Creation Toolkit Top-Down Example Agents

Furthermore, let us say that we set our world as follows: we create a 5 grid-space by 5 grid-space world, tile our Background Agent over the whole world, and then place a Box Agent as well as a Male and Female Rat Agent in the world. The following picture shows what this world might look like.

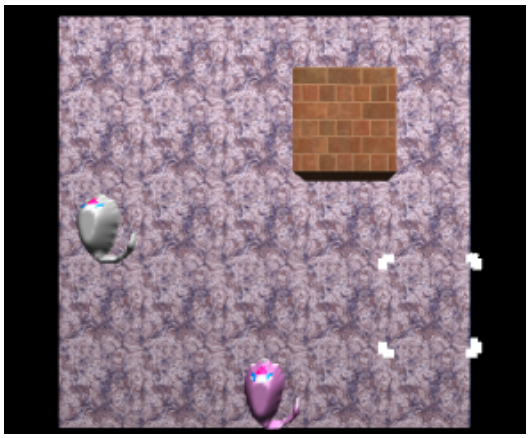


Figure 19: 5 By 5 World Setup for Simulation Creation Toolkit Top-Down Illustrative Example

At this point let us say the user wanted to make the Rat Agents move randomly using the Simulation Creation Toolkit. The user would first launch the system. This is accomplished by clicking on the "blue ball" icon button in AgentCubes (note that this button only exists in the Simulation Creation

Toolkit version of AgentCubes created for this thesis). The following picture depicts this button.

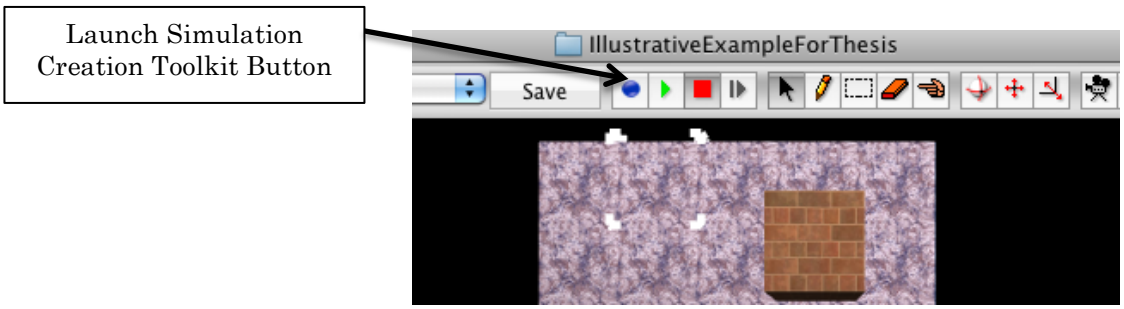


Figure 20: AgentCubes With Simulation Creation Toolkit Button

Clicking on the button depicted in Figure 20 launches the following window entitled "Simulation Construction Kit."

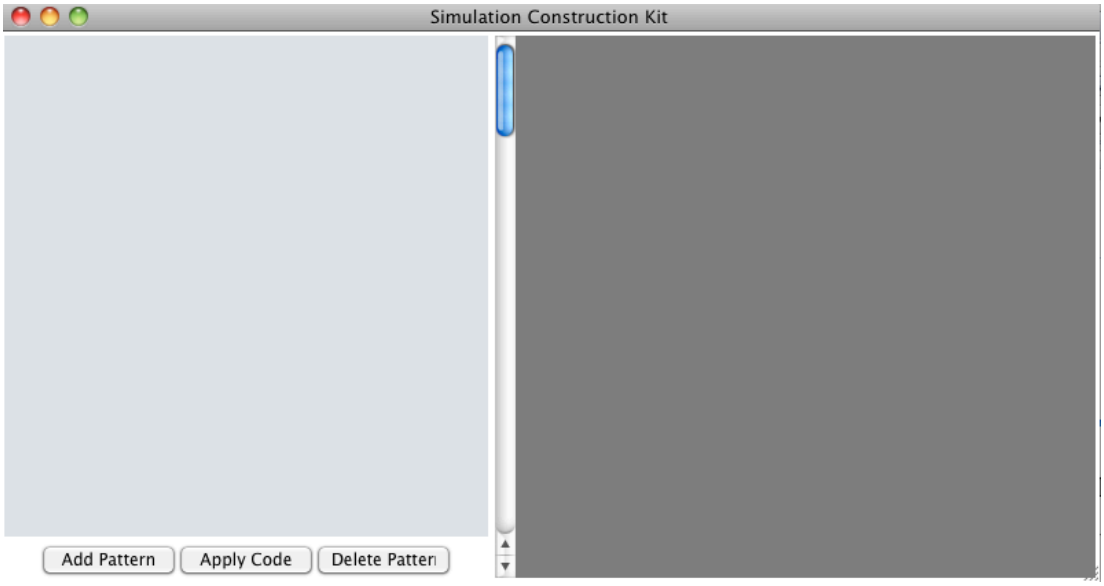


Figure 21: Simulation Construction Kit Window

The Simulation Construction Kit Window is split down the middle into a left and a right side. The left side keeps track of patterns implemented in the Simulation Creation Toolkit, and as will be shown, enables users to specify and re-specify any pattern they implement. The right side of the Simulation

Construction Kit Window displays interacticons corresponding to the pattern currently selected on the left side. There are three buttons at the bottom of this window (from left to right): “Add Pattern”, “Apply Code”, and “Delete Pattern”. The “Add Pattern” button launches a window that enables the user to select a pattern for their simulation. The “Apply Code” button does not currently serve any purpose; as we will see later, code is automatically and immediately applied and updated in AgentCubes upon pattern completion and specification. The “Apply Code” button should be removed in future versions of the system. Finally, the “Delete Pattern” button allows a user to remove a selected pattern along with its corresponding AgentCubes code. Since we want to apply the Random Movement Pattern to our Rat Agents, we would click on the “Add Pattern” button. Clicking on this button brings up the following window.

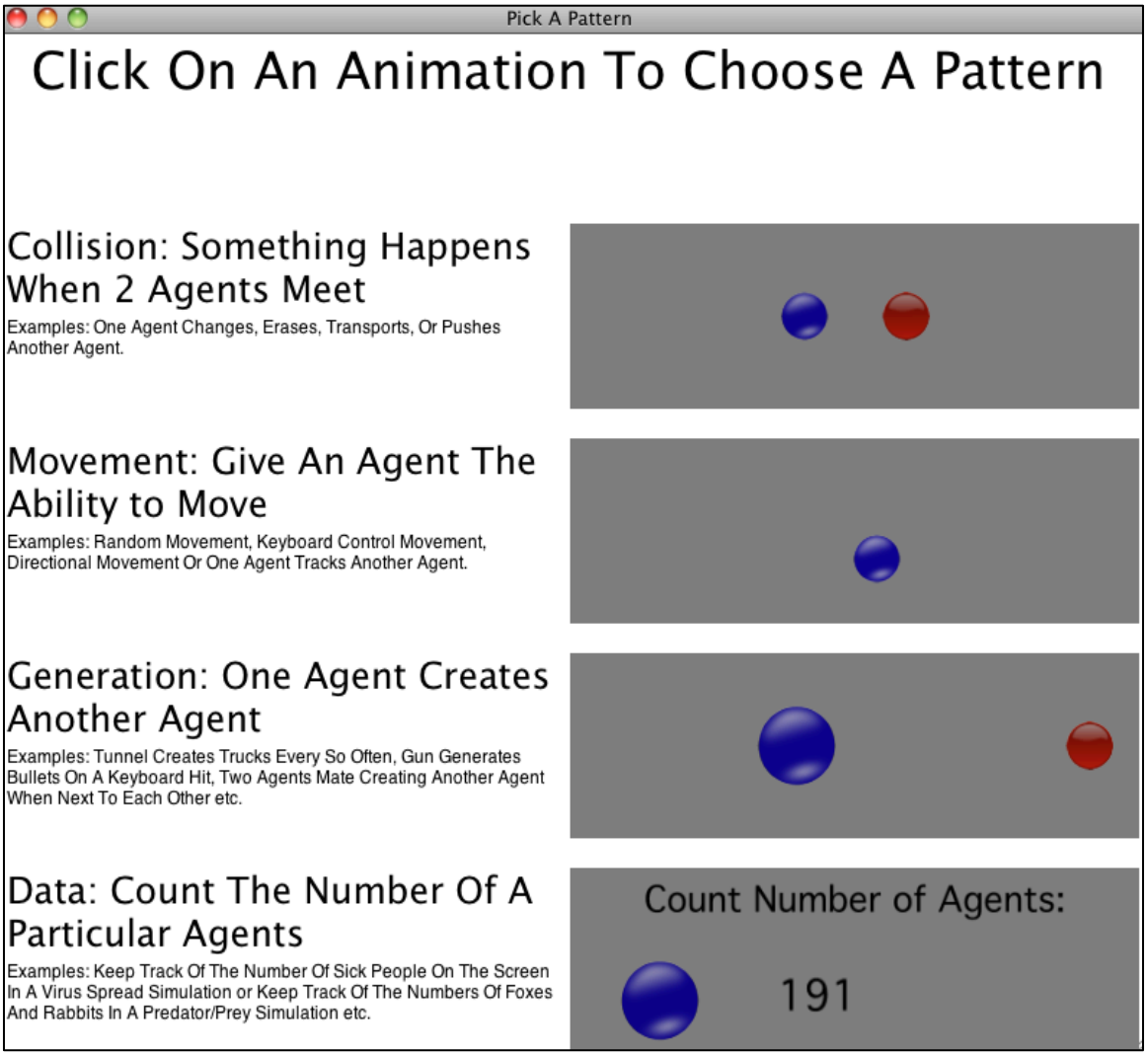


Figure 22: The Pattern Picker Window

Figure 22 depicts the "Pattern Picker Window". The left column of the window describes the pattern categories followed by examples of patterns or interaction contained within these categories. The right column of the window contains a representative interacticon for that particular category (see Appendix B). The first two categories from the top, Collision and Movement, each contain four patterns within them; the last two categories, Generate and Data Patterns, are their own category (see section 3.2). To open either the Movement and Collision subcategory windows or the Generation or Data Pattern windows, the user must click on the corresponding animation

as stated at the top. Since we are trying to add a Random Movement Pattern to our simulation, we will click on the animation corresponding to the Movement category in the Pattern Picker Window. This brings up the following Movement Picker Window.

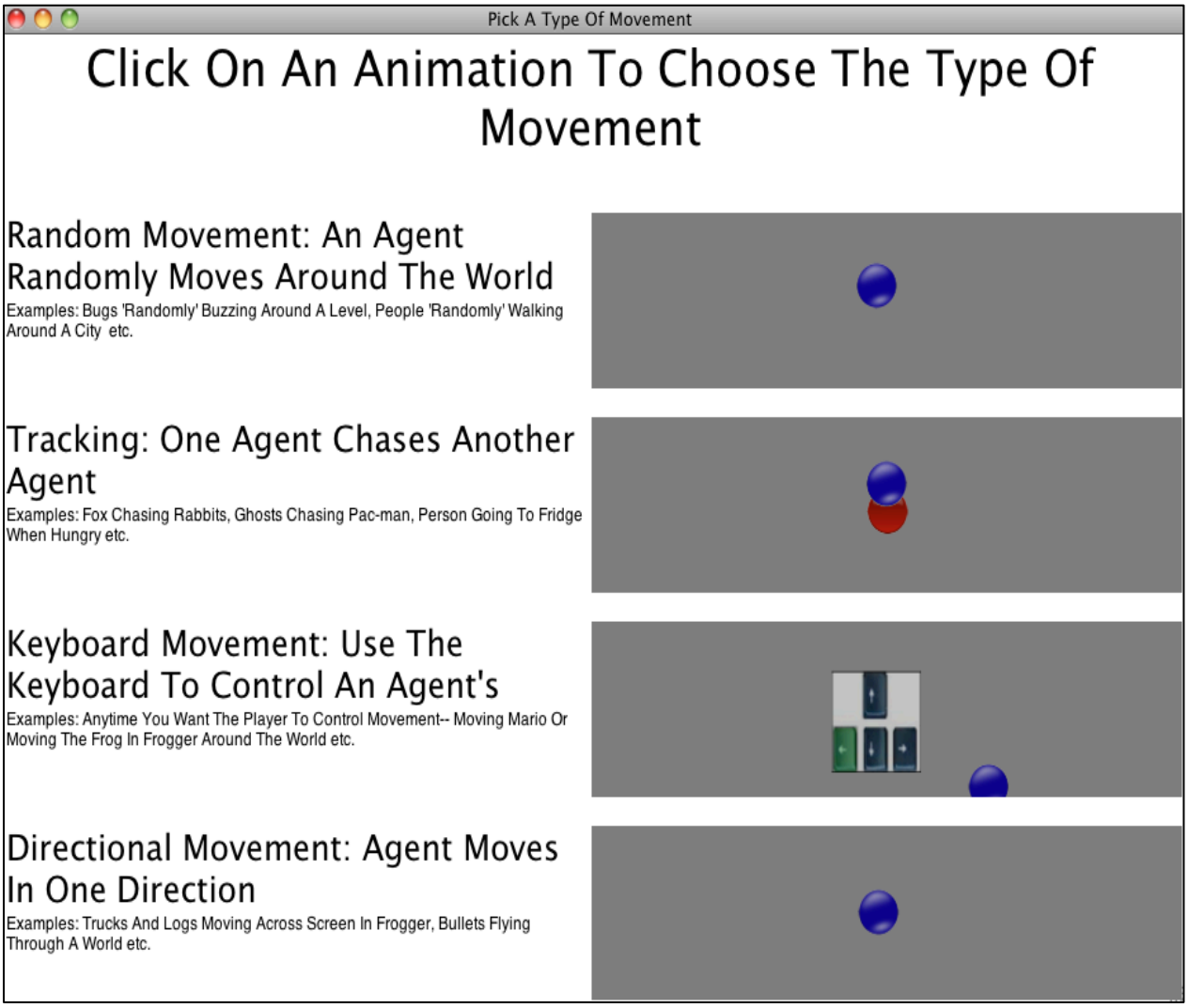


Figure 23: The Movement Picker Window

The setup of the Movement Picker Window is similar to the Pattern Picker Window (Figure 22), in that the left side contains pattern descriptions and the right side contains animated interacticons corresponding to the described

pattern on the left. Clicking on any of these animated interacticons will bring up the pattern window for that particular pattern. We will click on the first interacticon from the top in the Movement Picker Window to add the Random Movement Pattern to our simulation. Doing this brings up the following Random Movement Pattern Window.

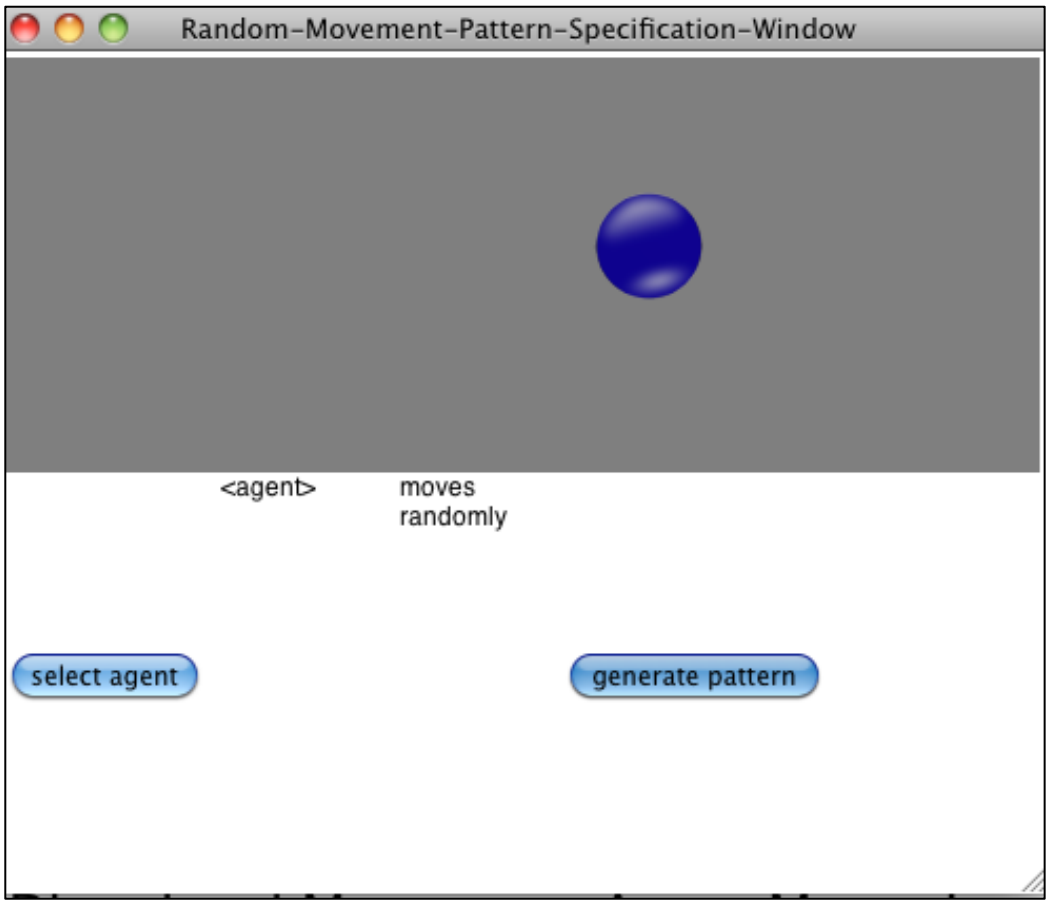


Figure 24: Random Movement Pattern Window

We see a few things apparent in Figure 24. The first is the interacticon consisting of a generic blue circle in the grey section located at the top of the window (see Appendix B.9). This blue circle moves around randomly exhibiting the Random Movement Pattern interacticon. Right under this window we see some text that says "<agent> moves randomly."

Under this sentence we see a button that says “select agent” to the bottom left and “generate pattern” to the bottom right. Clicking on the “select agent” button brings up a palette of every agent shape that currently exists in the simulation. The following picture shows the window with this palette opened.

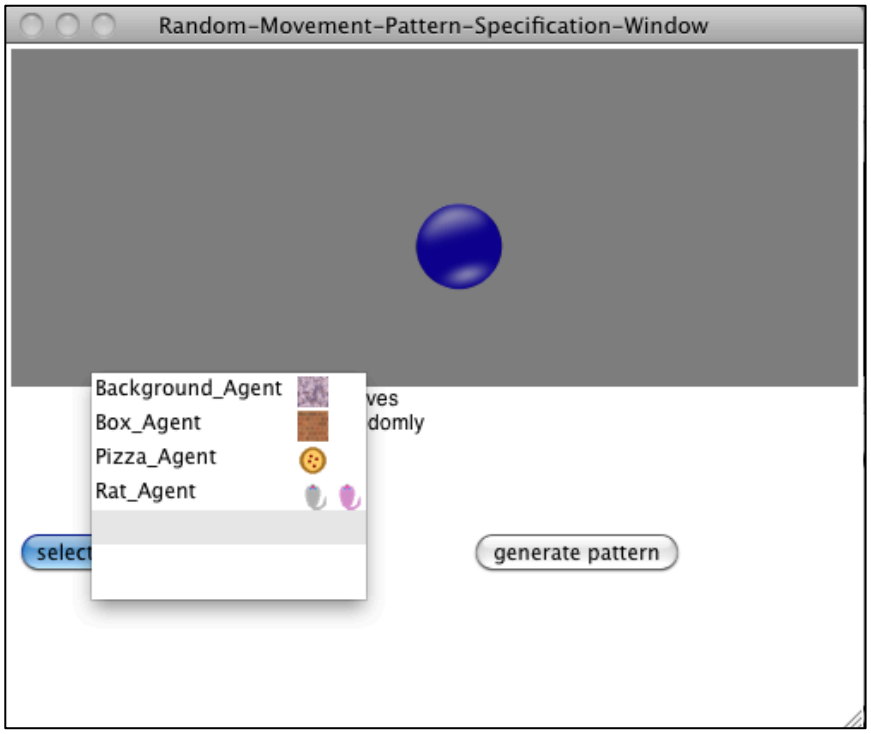


Figure 25: Random Movement Pattern Window With Shapes Palette

Since we want to apply this pattern to the Rat Agents, we will select one of the Rat Agent Shapes. The following picture depicts the selection of the Male (grey) Rat Agent.

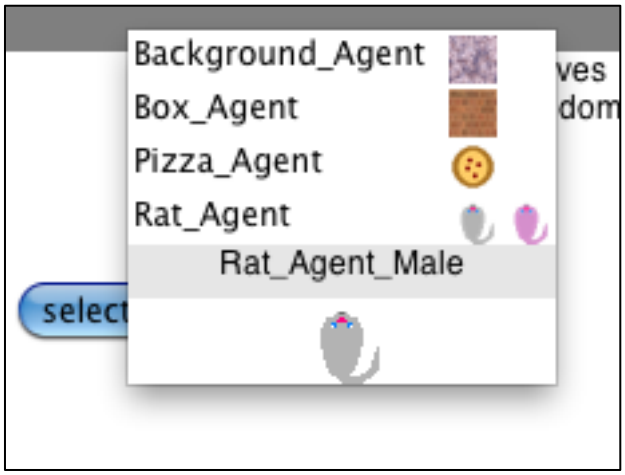


Figure 26: The Shapes Palette With The Male Rat Agent Selected

Selecting this shape changes the window to look like the following.

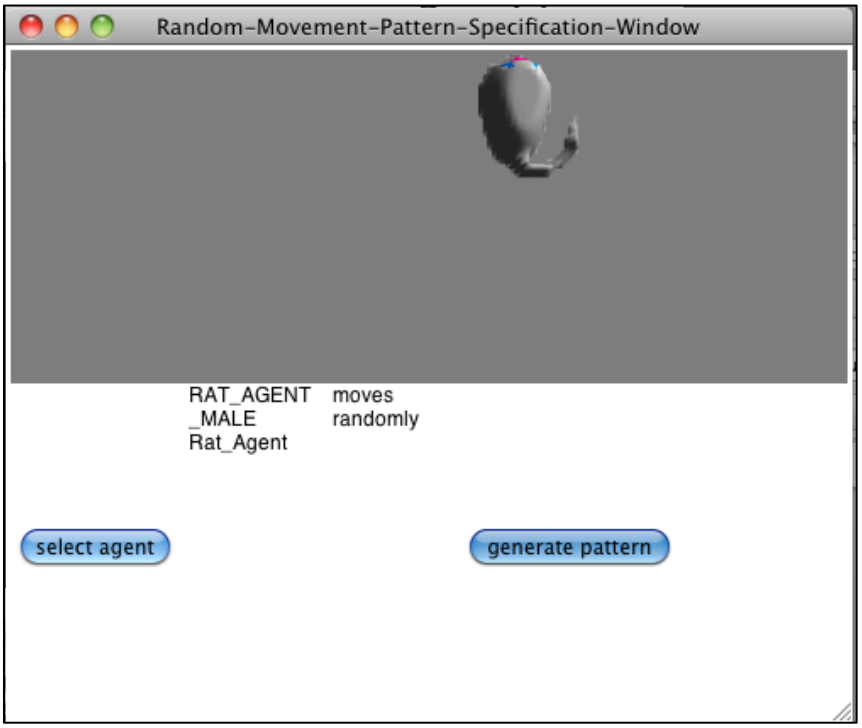


Figure 27: The Random Movement Pattern Window With The Male Rat Shape Selected

Two things have changed about this window. First off, the interacticon in the grey box at the top of the window now displays the Male Rat Agent moving

randomly instead of the generic blue disk. Secondly, the sentence under the interacticon replaces the "<agent>" string with the name of the shape and the agent, yielding the following sentence: "RAT_AGENT_MALE Rat_Agent moves randomly." The all caps word of the agent part of this string refers to the agent shape ("RAT_AGENT_MALE") and the lowercase part is the name of the agent ("Rat_Agent"). Similarly, if we had selected the Female Rat Agent, the full string would read "RAT_AGENT_FEMALE Rat_Agent moves randomly."

We are now ready to add the Random Movement Pattern to our simulation. We do this by hitting the "generate pattern" button located at the bottom right of the Random Movement Pattern Window (see Figure 27). The first thing that occurs after this button is hit is that all of the windows we may have opened on the way to selecting this pattern are closed leaving only the Simulation Construction Kit Window (originally depicted in Figure 21). Furthermore, the Simulation Construction Kit Window is now modified as shown in the following picture.

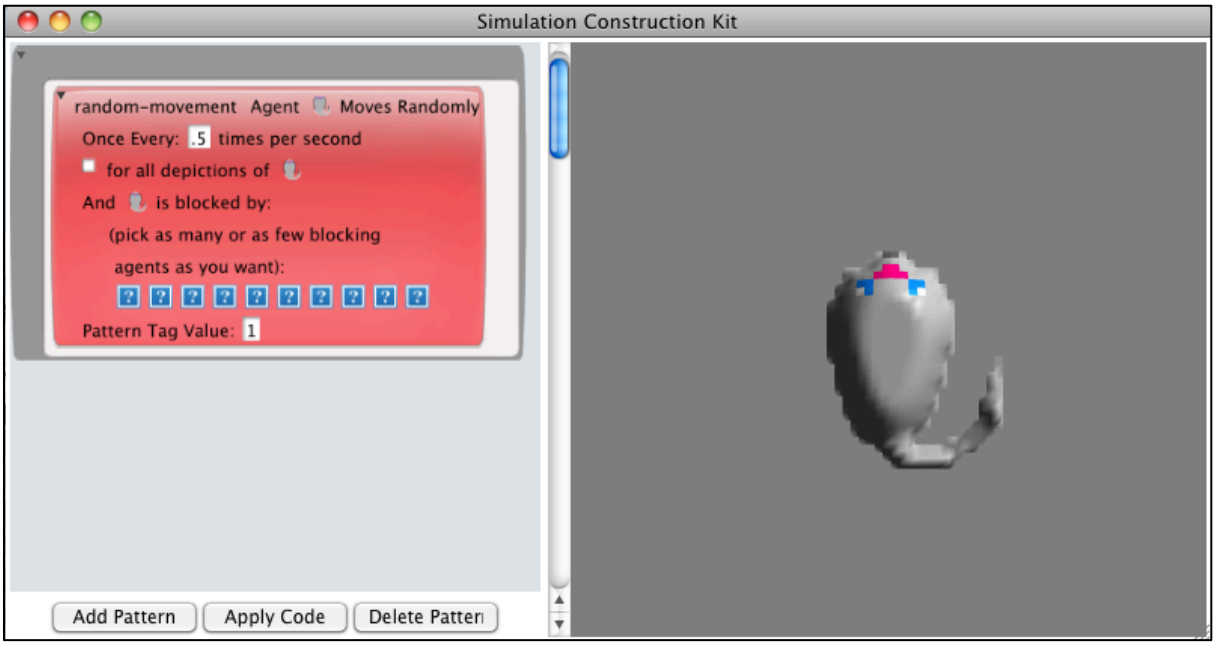


Figure 28: Simulation Construction Kit Window Post Random Movement Pattern Addition

There are a few notable things in Figure 28. First off, The Simulation Construction Kit Window now has a box contained with in its left half; this is called the "Random Movement Pattern Specification Box." The Random Movement Pattern Specification Box, similar to any pattern specification box that has been recently generated, starts selected (denoted by the red coloring). When a pattern's specification box is selected, the right side of The Simulation Construction Kit Window shows the interacticon for that pattern with the currently selected agent(s) for that particular pattern. Currently, the right half of the window has the Random Movement Pattern interacticon with the Male Rat Agent moving randomly.

3.3.1 In-Depth look at the Random Movement Specification

Now let us take a closer look at the Random Movement Pattern Specification Box.

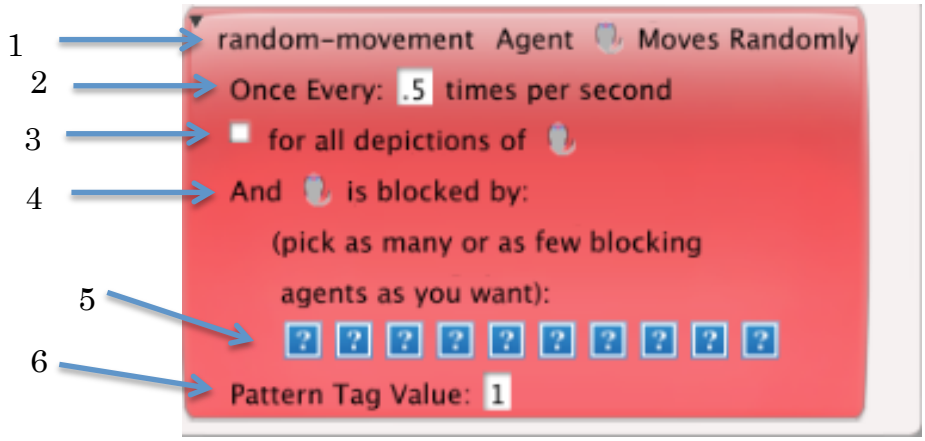


Figure 29: Close-Up Of The Random Pattern Movement Specification Box

Figure 29 labels each part of the Random Movement Pattern Specification Box. We will first introduce each label, and then proceed to describe the labels more in-depth. Label **1** specifies what agent is involved in the Random Movement pattern. In this case it is the Rat_Agent_Male Shape (the grey rat). Label **2** specifies how often this agent moves; in this case, the agent moves once every half second. Label **3** allows a user to hit a checkbox to specify whether or not this pattern is applied to just this specific agent's shape or all the shapes belonging to this agent. Labels **4** and **5** enable a user to select 10 different agent-shapes that block this Random Movement. Finally, label **6** is a Pattern Tag Value, which is a unique identifier for each implemented pattern that is automatically generated and used by the Simulation Creation Toolkit to identify the corresponding AgentCubes code for this pattern.

Label **1** in Figure 29 enables the user to re-specify which agent randomly moves. Let us say, for example, that the user actually wanted the Female Rat Agent Shape to move randomly. Instead of having to delete and re-implement the pattern, the user would instead click on the shape picture

in Label **1** to bring up a pop-up menu with all the agent shapes. This is depicted in the following.

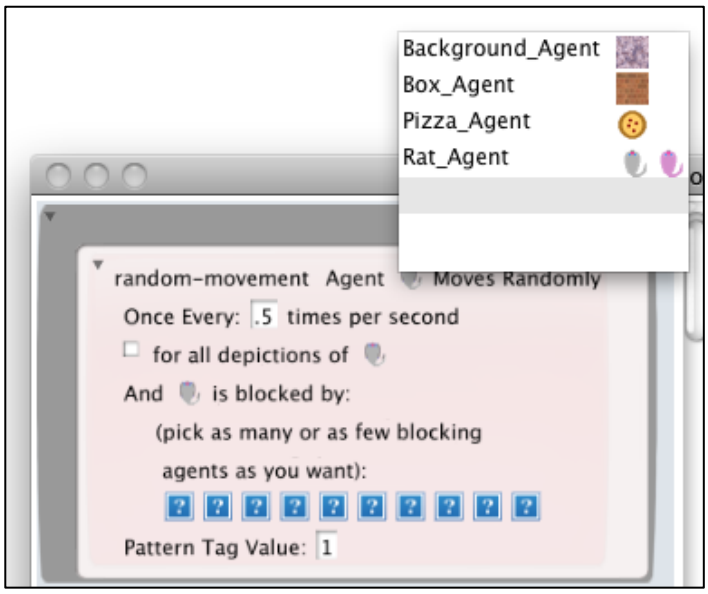


Figure 30: Random Movement Specification With Agent Pop-Up Menu

Let us say at this point we were to re-specify the Female Rat Agent Shape as the shape that moves randomly by clicking on that shape in the pop-up menu depicted in Figure 30. We would get the following picture.

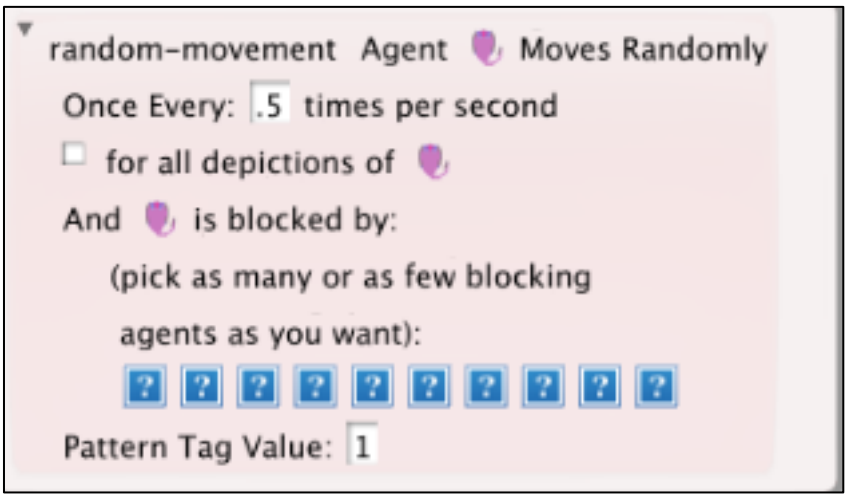


Figure 31: The Random Pattern Specification With The Female Rat Shape Selected

There are a few things of note in Figure 31. First of all, the specifications that relate to the selected shape in the Random Movement Pattern automatically change to the Female Rat Shape. These include labels **3** and **4** in Figure 29, which specify if this pattern affects all depictions of the randomly moving agent and what shapes block the random movement, respectively. Another thing to note, is that the shapes corresponding to **3** and **4** in Figure 29 are un-selectable and can only be changed when the user selects a given shape to apply the pattern to. The automatic changing of shapes that depend on the agent or agents enacting the pattern occurs in all the pattern specifications.

Label **2** in Figure 29 allows to the user to change the speed of the randomly moving agent. The user, for example, can change 0.5 to 1 if the user instead wants a rat that randomly moves once every second.

Label **3** in Figure 29 enables the user to apply this pattern to every shape corresponding to this agent by selecting a checkbox. In our case we want both the Female and the Male Rat shapes to move randomly. Therefore, for our purposes, we would select this checkbox. This would give us the following picture for our pattern specification (note that we have changed back to the Male Rat Shape to be consistent with Figure 29 though its unnecessary since we are applying the pattern to every shape of the Rat Agent).

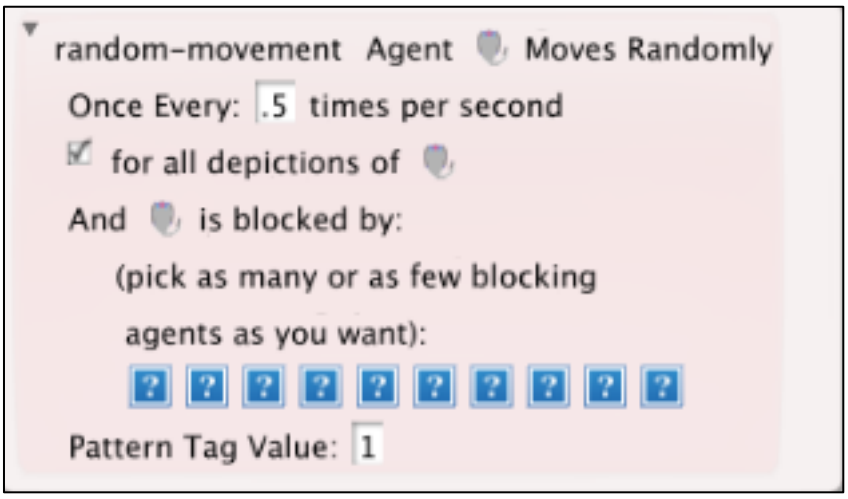


Figure 32: Random Movement Specification With All Depictions Checkbox Selected

Labels **4** and **5** in Figure 29 enable the user to select up to 10 shapes to block the random movement of the selected agent by clicking on a question mark box. Each one of the question mark boxes can be changed to a shape present in the simulation. The following figure depicts the pop-up menu that is shown when a question mark box is selected.

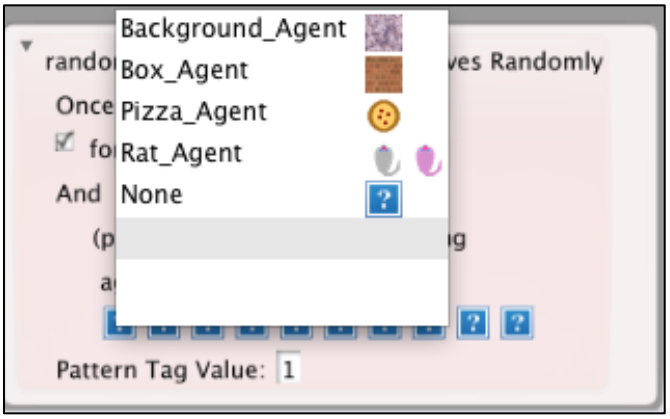


Figure 33: The Blocking Agents Pop-Up Menu For The Random Movement Pattern Specification

The pop-up menu in Figure 33 is very similar to the previous shape palette menus presented in Figure 30 and Figure 26 with one key difference. The shape palette menu in Figure 33 contains the choice of a Question Mark

Shape entitled “None” in addition to all the agent-shapes contained within the simulation. This is included because if a user selects a shape as a blocking agent and later, decides that she/he does not actually want to have this shape block the movement, the user can change that choice back to a Question Mark Shape without having to re-implement the pattern. For illustration purposes, let us say we select the Pizza_Agent as a blocking shape. The final picture, after we have made all of our changes to the Random Movement Specification, would look as follows.

Figure 34: Final Random Movement Pattern Specification

At this point, the Random Movement Pattern implementation is complete, and if we run the simulation, the Rat Agents randomly move around our 5 by 5 world depicted in Figure 19. Furthermore, if we place any Pizza Agent instances in the world, neither Rat Agent moves onto it; however, the Rat Agents do move onto the Box Agent.

Finally, label **6** in Figure 29 specifies a Pattern Tag Value for this pattern. This is not something the user can change, but rather, is automatically generated by the Simulation Creation Toolkit in order to match AgentCubes code with its corresponding pattern. To get a better

understanding of this concept, as well as an idea as to what these previous steps accomplished, it helps to look at how the Simulation Creation Toolkit handles the pattern implementation in AgentCubes (i.e. what behaviors are added to the agent) at each stage of this specification.

3.3.2 In-Depth look at the Random Movement Implementation

We will now look at what code is added to AgentCubes through each step of section 3.3.1. We will also discuss the general way the Simulation Creation Toolkit implements patterns and deals with timing. Before implementing any patterns, the Rat Agent has no rules in its behavior as shown by the following picture which depicts an empty While-Running method.

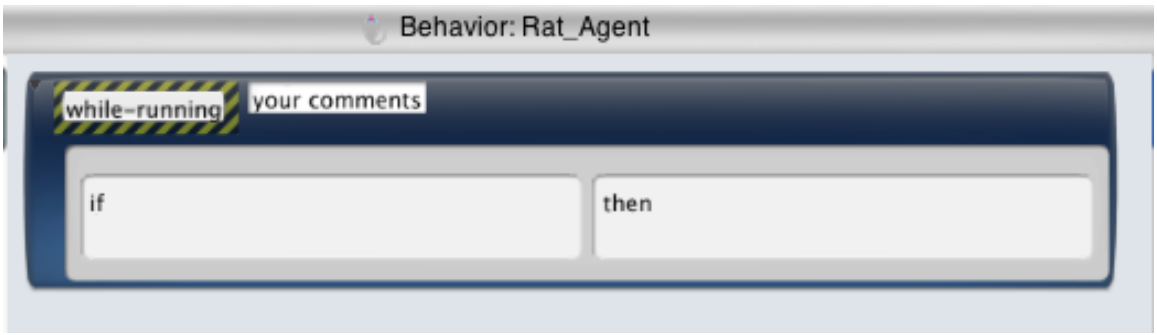


Figure 35: The Rat Agent Behavior With No Rules

As we go through the steps of 3.3.1, the first place the Simulation Creation Toolkit automatically adds rules to the Rat Agent Behavior, occurs when we hit the "generate pattern" button in Figure 27 yielding Figure 28. After this step the Rat Agent behavior looks as follows.

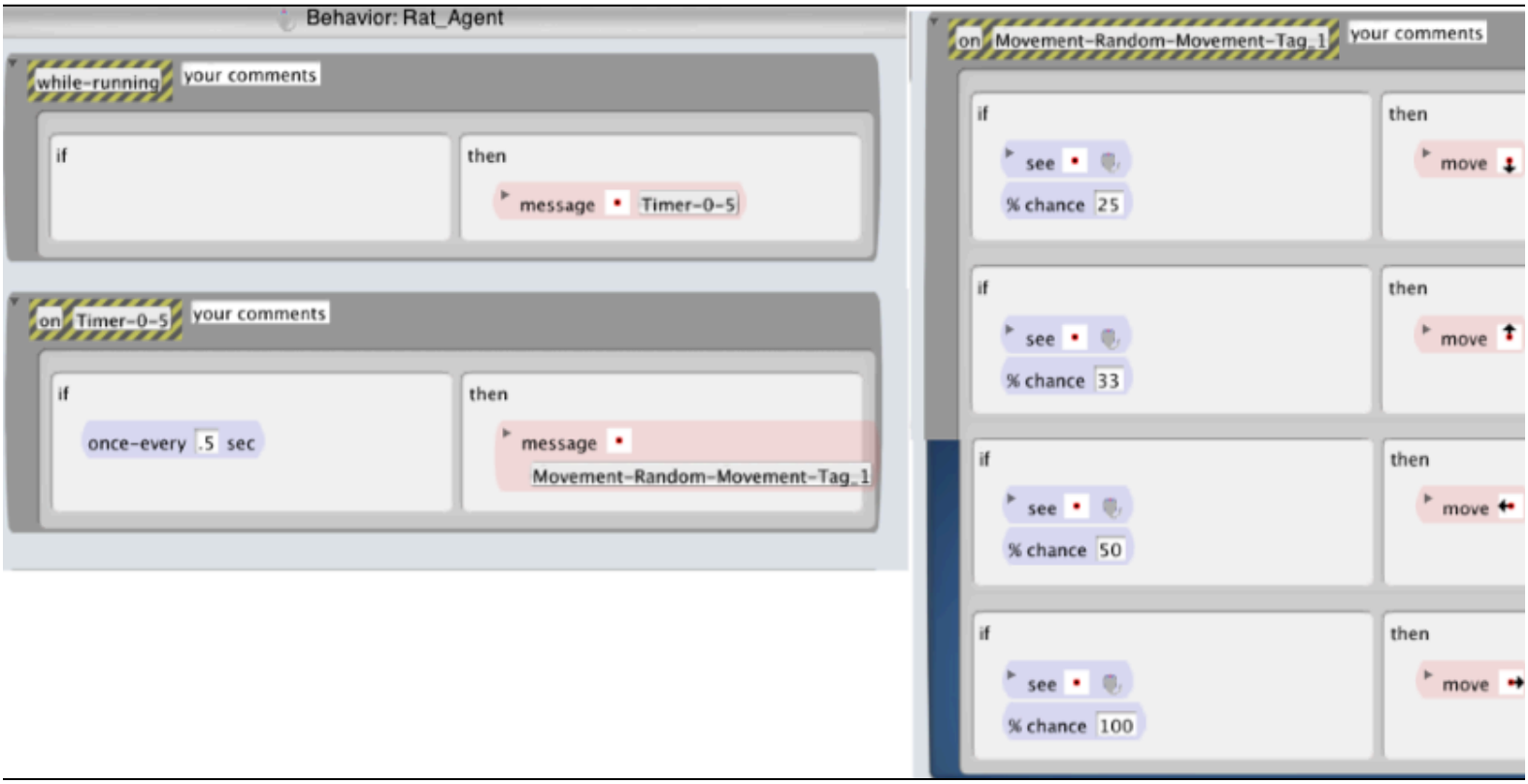


Figure 36: Rat Agent Behavior After The Addition Of The Random Movement Pattern Depicted In Figure 28

Figure 36 has a lot of information in it. First off, the top left corner of Figure 36 depicts the now modified While-Running method of the Rat Agent behavior. In AgentCubes, the While-Running method is the only method guaranteed to be called each agent update cycle. This method now contains one rule. This rule contains one action and no conditions. Therefore, at present time, this rule will be executed every agent update cycle because there are no conditions to be met. The **message action** contained within this

rule passes a message to the Rat Agent itself to call a method named "Timer-0-5."

Right under the "While-Running" method in Figure 36 is the method named "Timer 0-5". This method has one rule containing a single **once-every condition** and a **message action**. The condition states that once every 0.5 seconds the action will happen. The action of this rule is another message to the Rat Agent itself calling a method named "Movement-Random-Movement-Tag_1." Finally, the right side of Figure 36 depicts this "Movement-Random-Movement-Tag_1" method. This method's code contains the rules that actually move the Rat Agent. We will generally refer to methods that specifically enact the pattern as "pattern methods" for the remainder of this chapter. Patterns may add multiple pattern methods among many different agents; this is common in patterns where two agents interact with one another for example.

Note that pattern methods have the Tag Value appended to its method name; in this case "Tag_1" is appended to a string that denotes the type of pattern implemented, "Movement-Random-Movement." As mentioned in the previous section, each pattern has a unique Tag Value. This value is determined by incrementing the previous Tag Value as each pattern is implemented in the Simulation Creation Toolkit. Therefore, if another Random Movement Pattern were to be added to the simulation, it would be named "Movement-Random-Movement-Tag_2." Similarly if we then added a Generate Pattern to the simulation, it would be named "Generate-Pattern-Tag_3" etc. This Tag Value appended to the method name matches the Tag Value in the patterns specification box. For example, in Figure 29 we see that the Random Movement Pattern Specifications Box has a Tag Value of 1 too. Therefore, the Random Movement Specification in Figure 29 corresponds to

the pattern method in Figure 36. In patterns wherein two agents each have a pattern method, the Tag Value is still the same for both pattern methods.

The consistency between Tag Values among implemented patterns in the Simulation Construction Kit Window and the corresponding pattern method code in AgentCubes, enables the Simulation Creation Toolkit to modify and delete previously implemented patterns. When a pattern specification is updated, the Simulation Creation Toolkit can find the correct method to alter by matching the pattern method name Tag Value in a particular agent with the pattern specification Tag Value. Furthermore, if a pattern is deleted in the Simulation Construction Kit Window, the Simulation Creation Toolkit can delete not only the pattern method(s) based on the Tag Value, but also, any calls made to the method(s) by searching for this Tag Value in **message actions**, for example.

In the Simulation Creation Toolkit, every pattern call (i.e. the **message action** that invokes a pattern method) is put into a method that reflects its timing. These will be referred to as "timer methods" for the remainder of this chapter. In this case, the default specification for the Random Movement Pattern is once-every 0.5 seconds (see label **2** in Figure 29). The Simulation Creation Toolkit starts adding AgentCubes code immediately after a pattern is generated; when this pattern is implemented in AgentCubes, the Simulation Creation Toolkit makes a method called "Timer-0-5" referring to the fact that this pattern should occur once every 0.5 seconds. The reason the method is called "Timer-0-5" and not "Timer-0.5" is that AgentCubes does not allow the period character in method names, so a dash is used instead to demarcate the whole number portion of the time from the fractional portion of the time. Thought of another way, if we had a

pattern that occurred once every 15.3 seconds, the Simulation Creation Toolkit would make a method called "Timer-15-3."

Timer methods serve an important role in the implementation and execution of patterns. In order to fully understand the code created by the Simulation Creation Toolkit, as well the potential impact of certain pattern specification choices, the concept of timer methods must be clear. Thus, before we continue looking at the Random Movement Pattern implementation, we will explain how timing works in the Simulation Creation Toolkit.

3.3.3 Timer Methods In The Simulation Creation Toolkit

All of the Simulation Creation Toolkit timer methods have only one rule in them. The condition of this rule is always a single **once-every condition** that reflects the timing of the method. In this case, since we move once every 0.5 seconds, the Simulation Creation Toolkit creates a "Timer-0-5" method with a **once-every condition** set to 0.5. Similarly, if we had a "Timer-15-3" method, it would contain one rule with a **once-every condition** set to 15.3. It should be noted that patterns that occur "instantaneously" are put in a "Timer-0-0" method.

Any Simulation Creation Toolkit timer method has one rule with one condition but may have multiple actions. Every single one of these actions will be a **message action** that calls a particular pattern. One can think of a timer method as a bucket that contains all the patterns that need to be executed at a particular time. In our case, let us say the Rat Agent, in addition to moving randomly once every 0.5 seconds, also generated something (i.e. the Generate Pattern) once every 0.5 seconds. Then our "Timer-0-5" method would look as follows.

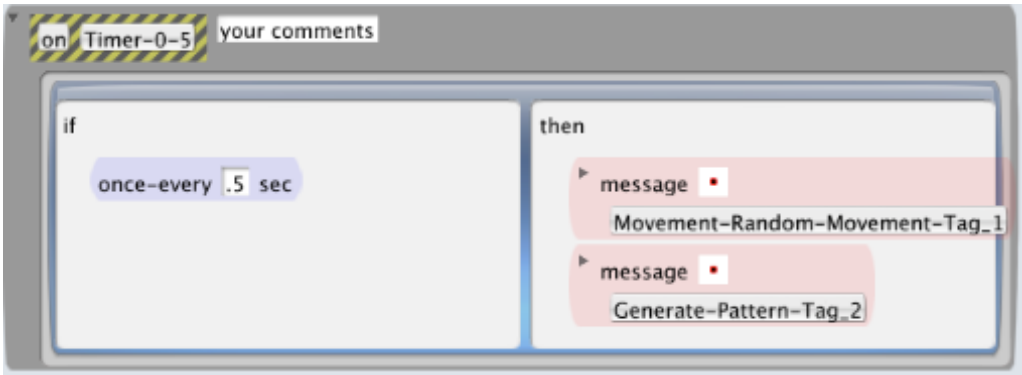


Figure 37: "Timer-0-5" Method With Random Movement And Generate Pattern Message Actions

The Simulation Creation Toolkit puts messages to pattern methods that are executed at the same time in the same timer method. In Figure 37 we also see that the Generate Pattern is given a Tag Value of 2. When we delete this Generate Pattern in the Simulation Construction Kit Window this **message action** contained within the Timer method is also deleted in addition to the method entitled "Generate-Pattern-Tag_2." Furthermore, if this **message action** happens to be the last action in the timer method, the timer method itself is deleted and the **message action** call in the While Running method to that particular timer is also deleted.

Timer methods add an extra step that may initially seem unnecessary for the implementation of patterns. However, they are integral in enabling simulations created using the Simulation Creation Toolkit to run correctly. The following examples outline why.

One might think that instead of adding these timer methods, we could instead put all the Simulation Creation Toolkit pattern method calls in the While-Running method itself; for example, we could use individual rules as the "timer buckets" instead of timer methods. Let us say we had a Generate Pattern being executed once every 0.5 seconds, a Random

Movement Pattern being executed once every 0.7 seconds and an Absorb Pattern being executed once every 0 seconds. Using individual rules as our "timer buckets", we would expect three rules each with a different **once-every condition**. The While-Running method might look as follows.

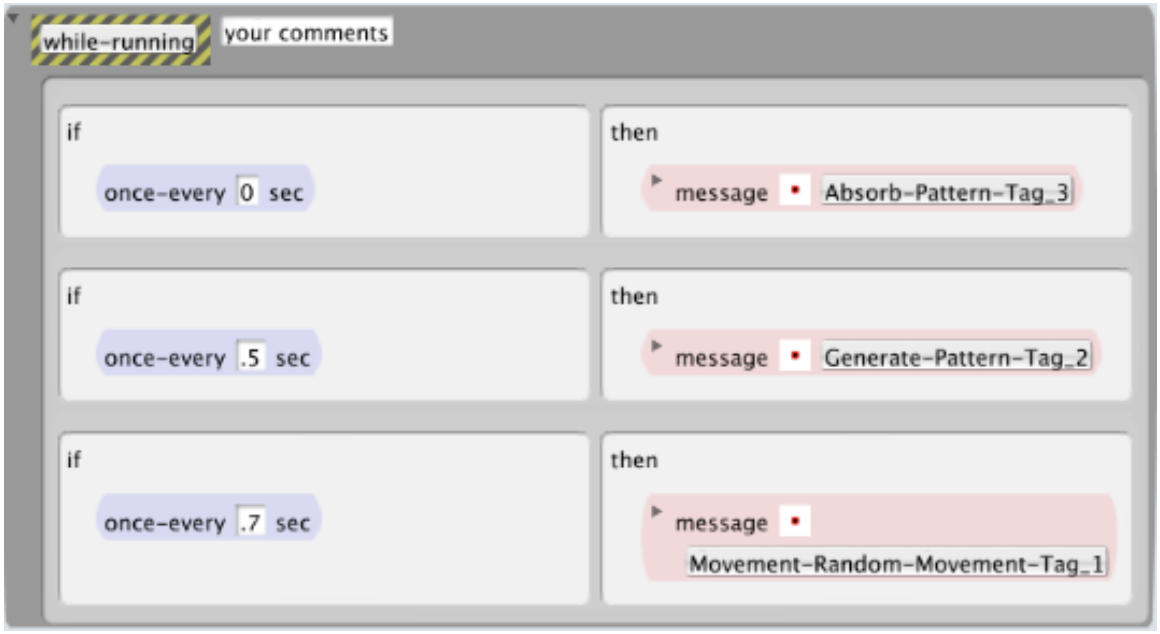


Figure 38: An Example Of What Pattern Calls From A While-Running Method Might Look Like

There are two factors at play that describe why Figure 38 will not work as the user intended. The first is, AgentCubes goes through every update cycle and executes at most one rule in the While-Running method of each agent (it might execute zero rules if no conditions are met). Recall that the While-Running method is the only method guaranteed to be called every agent update cycle. Furthermore, AgentCubes checks method rules from top to bottom. Note that this does not mean only one rule is executed each update cycle for a particular agent because the While-Running method rule that is executed might make a call to another method contained within the agent's

behaviors. However, AgentCubes will only execute at most one rule in any method called each update cycle.

The second factor is that a **once-every condition** is met if the time between the last time that particular rule was executed and the current update cycle is greater or equal to the time specified in that particular **once-every condition**. Therefore, in the case of Figure 38, only the Absorb Pattern rule will ever fire because the time between the last time that rule executed and the current update cycle is always going to be greater than 0 seconds. The Generate Pattern and the Random Movement Pattern will never execute during this agent's update cycle and the program will probably not perform as the user expected (considering they implemented all 3 patterns for this agent).

The fix for the problem displayed in Figure 38 is to organize all the rules in descending order of time. The following figure depicts how we would accomplish this.

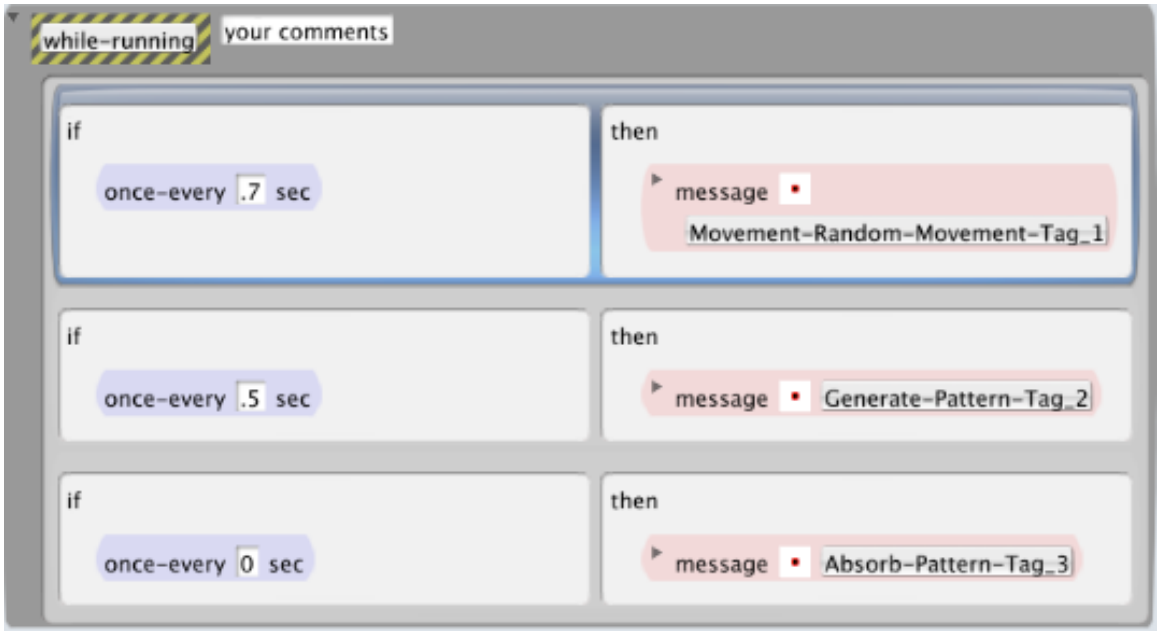


Figure 39: The Rules From Figure 38 Re-Organized In Descending Order Of Once Every Conditions

One can imagine that this would work more effectively than Figure 38. The first rule (Random Movement) would execute whenever 0.7 seconds passed between its last execution. The second rule (Generate) would execute anytime the first rule did not execute, and yet, 0.5 seconds had passed since the last time this rule executed. If neither the first or second rule executed, the final rule (Absorb) would execute. Hypothetically, given a small amount of time between update cycles, every pattern method would have a chance to be called in this scenario.

Organizing pattern method calls by descending **once-every condition** time in the While-Running method was the original way the Simulation Creation Toolkit added patterns to agents, and in small-scale simulations, this is an effective solution. However, let us say we had an AgentCubes simulation with a large number of agent instances in a world. As the number of agents present in a world increases, so does the time between update cycles for a given agent instance in the world. If we keep on

increasing the number of agents in the world and run the simulation, at some point, the time between update cycles for a given agent will eventually exceed 0.5 seconds. Therefore, the third rule in the While-Running method in Figure 39, referring to the Absorb pattern, will never execute.

Now let us say we continue slowly increasing the number of agents in the world; similarly, at some point we will have so many agents that the time between update cycles for a given agent will exceed 0.7 seconds. Therefore, at this point our second rule, referring to the Generate Pattern, and our third rule, referring to the Absorb Pattern, will never execute. As our simulation increases in size, or our timer times get closer and closer together, eventually we will hit a point wherein the strategy outlined in Figure 39 will suffer from the same exact problems experienced in Figure 38.

Let us now look at how the Simulation Creation Toolkit would implement the above example. First off, the While-Running would only have calls to timer methods in a rule with no condition. This might look as follows.

Figure 40: The Simulation Creation Toolkit Organizing 3 Times In The 'While Running" method

The first thing to notice in Figure 40 is that all three timer methods are called every agent update cycle. Furthermore, this also means that the order

in which the timer methods are called does not matter. The three timer methods would look as follows.

Figure 41: The Timer Methods Called In Figure 40

Based on the While-Running method depicted in Figure 40, every method in Figure 41 should be called during each update cycle for this agent. At this point, each individual method checks what time has elapsed since the last time it was executed, and if the time has exceeded that amount, it will execute that particular pattern. Therefore, every pattern that the user implements will have a chance to be executed using this strategy, and no pattern method should ever not have a chance to be executed during a simulation run.

This strategy is not without problems either. For example, if the user wants a certain pattern to execute once every 0.7 seconds, this timing is not guaranteed once the simulation reaches a certain amount. This topic is actually very complex as it not only has to do with the size of the simulation, but also, the computer hardware currently being used to run the simulation. Regardless of these factors, as the number of agents in a simulation increases and the number of rules AgentCubes must check each update cycle increases, there will come a point where the time between an Agent's last update and current update greatly exceed the time specified for a pattern to be executed. In this case, the pattern will still execute, but not on time. However, the user intended for every pattern they implement to be executed at some point, and the timer method strategy employed in the Simulation Creation Toolkit gets closer to accomplishing this ideal.

The underlying reason as to why the Simulation Creation Toolkit must use Timer methods stems from the fact that the Simulation Creation Toolkit implements these patterns automatically in AgentCubes. When a user adds low-level rules in AgentCubes, the user also implicitly prioritizes these rules; for example, a rule at the top of the While-Running method would have preference over a rule at the bottom of the While-Running method. This is because, recall from above, methods only execute at most one rule every update cycle in AgentCubes. Therefore, if a rule is towards the top of a method, its condition gets checked first, and if it is deemed true, that action will get fired and no other rule will be executed in that method. Rules that comprise a method in AgentCubes can be thought rule 1 of the method will be executed, or instead, rule 2 of the method will be executed etc.

In AgentCubes the user is able to drag and drop rules that they have created such that they are higher or lower in priority in a given method.

However, in the Simulation Creation Toolkit, rules and methods are added automatically. Furthermore, it is implied that each pattern will execute regardless of the prior or subsequent implemented patterns for a particular agent. By forcing each pattern to be checked, regardless of their time condition, enables each pattern implemented to have the same priority level. The timer method strategy employed by the Simulation Creation Toolkit can be thought of as pattern 1 will be checked for execution and pattern 2 will be checked for execution etc.

At this point, we will continue the discussion on how the Simulation Creation Toolkit implements the Random Movement Pattern accomplished in 3.3.1.

3.3.4 Back To The Implementation Of The Random Movement Pattern

Now let us look at the right side of Figure 36, namely the Movement-Random-Movement-Tag_1 method. Recall the method looks as follows.

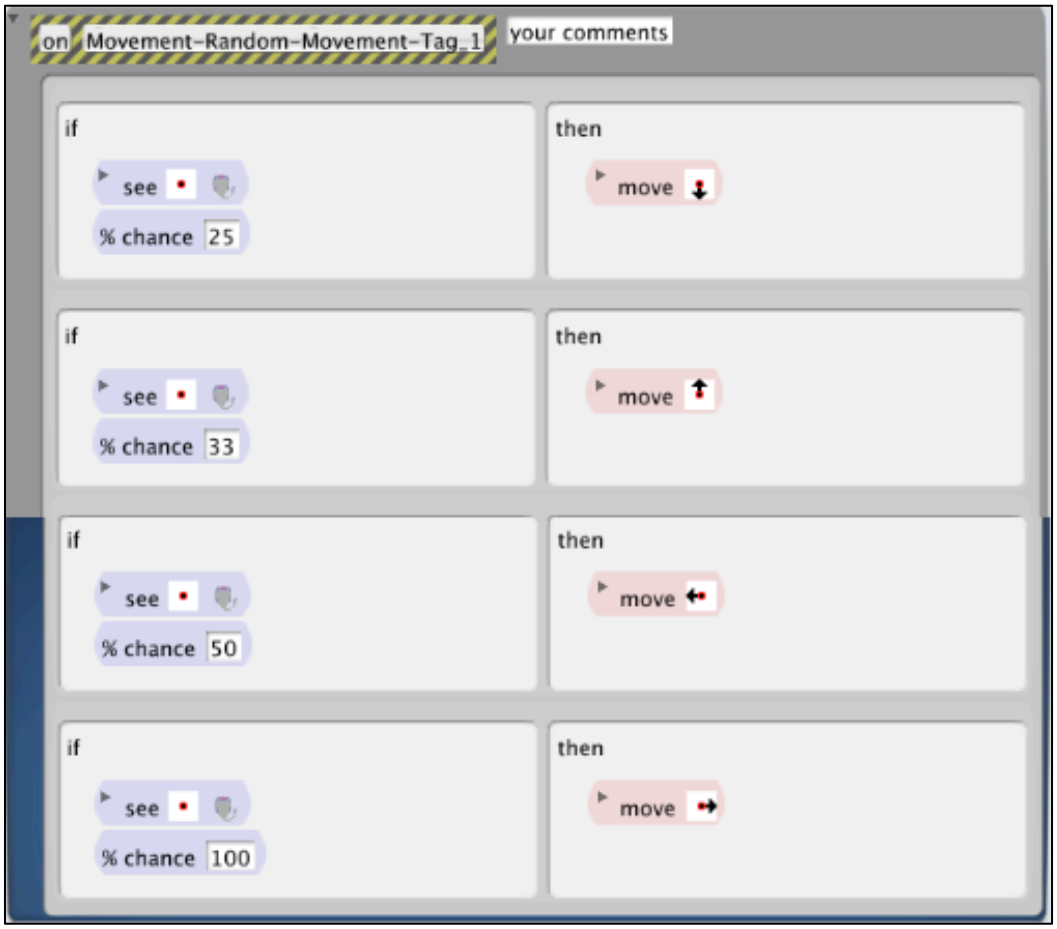


Figure 42: Movement-Random-Movement-Tag_1 Method

The method in Figure 42 accomplishes the Random Movement. Each rule states that if the current agent shape happens to be the Male Rat shape, with a given percent chance, this Rat Agent instance will move a given direction in the world. The difference in percent chance for each direction has to do with the rule's priority in AgentCubes (see section 3.3.3). Since rules are checked from the top, and the Rat Agent should have the same percent chance of moving in any direction, we need to make the percentages reflect those choices. When we check the top rule, that rule should have a 25% chance of being executed because there are four possible directions for the Rat Agent to move. If this rule does not get executed, the next rule has a 33% (really, it has a 1/3 or 33.33333…% of being executed but we shorten it to

33%; this should be corrected in subsequent versions of the Simulation Creation Toolkit) chance of being executed because there are only 3 directions left. Similarly, the second to last rule has a 50% chance of being executed, and if we get to the last rule, it should have a 100% chance to move in that direction.

A question arises as to why does the implementation not employ the **move-random action** or the **move-random-on action** instead of having one rule for each direction. There are two reasons for this. The first is that these AgentCubes actions enable you to move random on any agent or move randomly on one agent (presumably a background agent). The Simulation Creation Toolkit enables users implementing movement patterns to select any number of agents to move on or be blocked by; therefore, the agent needs the ability to check if one or more blocking agents are in a given direction before it moves in that direction. Splitting the **move actions** into different rules allows for this condition to be checked before moving a particular direction. The second reason is that modulating patterns rely on the ability to change the condition for each move rule to reflect that modulating pattern. For example, if something that the Rat Agent Transports is stacked on top of the Rat Agent, the Rat Agent cannot move in a given direction; it must instead transport in that direction. This will become clearer in the next section when we implement a modulating pattern.

Now, we want both the Male and Female Shape of the Rat Agent to move randomly. Recall from Figure 32 that the user accomplishes this by clicking the "for all depictions" checkbox in the Random Movement Pattern Specification. When the user clicks on this box, the Simulation Creation Toolkit immediately updates the Movement-Random-Movement-Tag_1 method to look as follows.

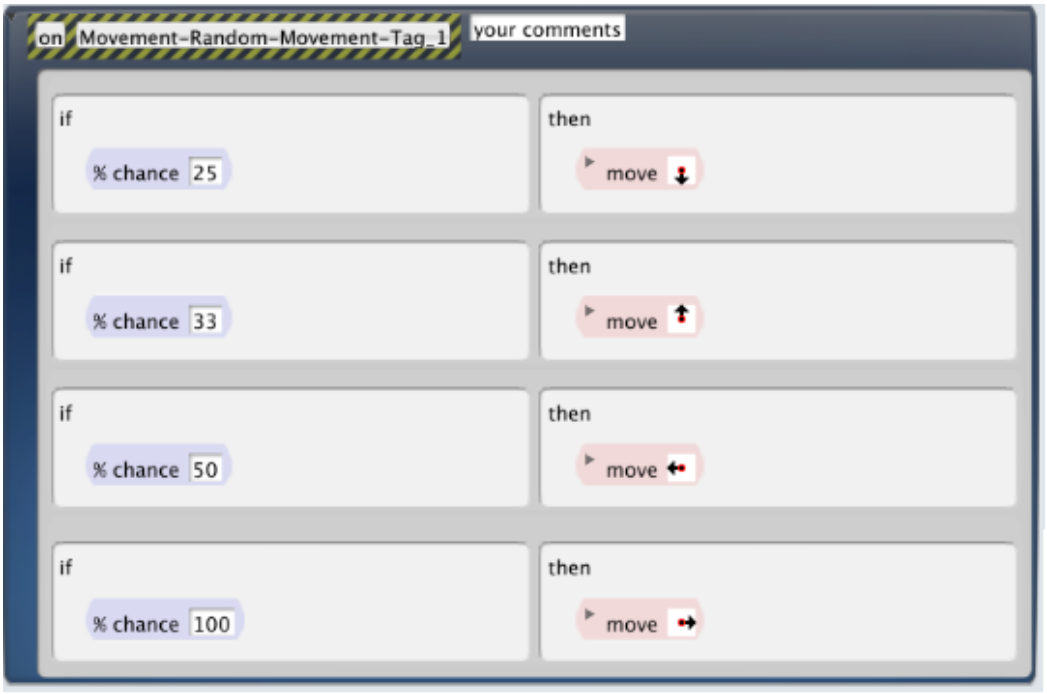


Figure 43: The Movement-Random-Movement-Tag_1 Method After "for all depictions" Is Specified

Notice that in Figure 43, the Rat Agent no longer checks to see if it is the Male Rat shape; regardless of its shape, the Rat Agent will now move randomly.

The final specification we make in Figure 34 is that the Random Movement is blocked by the Pizza Agent. When we make this change the Simulation Creation Toolkit automatically makes the following changes to the Movement-Random-Movement-Tag_1 method.

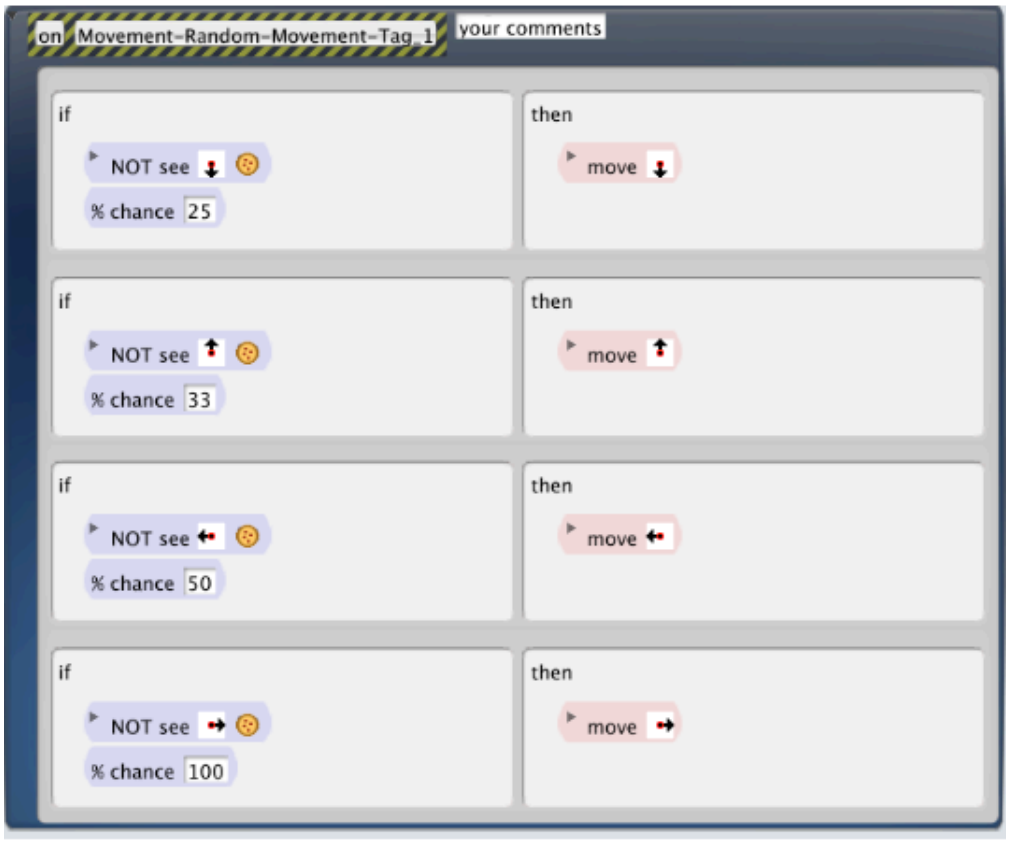


Figure 44: Final Rules For The "Movement-Random-Movement-Tag_1" Method

After we specify the Pizza Agent as a blocking agent, each rule in the method gets updated to reflect that if a Pizza Agent happens to be in a given direction the, Rat Agent will not move in that direction. This is accomplished by adding the **not see condition** to each rule. The Rat Agent, therefore, may not move at all in a given update cycle based on the percentages and the blocking agents around it; a more complicated solution would be to check every direction for a blocking agent, and then change the percentages based on the valid directions of movement for the Rat Agent. However, this is a suitable interpretation of the random movement pattern in that the Rat Agent will move randomly and will never actually move onto a Pizza Agent. If another blocking agent were to be specified, another **not see condition** with

that agent would be automatically added to each rule in Figure 44 by the Simulation Creation Toolkit.

At this point we have successfully made both Rat Agents move randomly and have seen how a user adds a pattern to the Simulation Creation Toolkit, specifies a given pattern, and how the Simulation Creation Toolkit adds rules to AgentCubes based on this pattern and its specifications. Most patterns in this toolkit follow this basic template with the exception of modulating patterns that are unique in the way that they modify other pattern's methods. Let us take a look at an illustrative example that depicts how the addition of a modulating pattern changes our currently implemented Random Movement Pattern.

3.4 Illustrative Example: How Modulating Patterns (Transport and Push) Are Implemented

Recall that modulating patterns effect the implementations of previously or subsequently applied patterns. There are two modulating patterns in the Simulation Creation Toolkit: Transport and Push. The implementation of these patterns does not change the behaviors of any agent unless a movement pattern is added to an agent before or after the addition of these modulating patterns. For example, if we have two agents in our simulation, Agent A and Agent B, implementing the Transport Pattern such that that Agent A transports Agent B, without any other patterns implemented, will yield a simulation wherein nothing happens. However, the moment we give Agent A the ability to move by implementing a Random Movement Pattern, for example, the Random Movement Pattern method will include rules wherein Agent A will only move a given direction if it is not

stacked under Agent B, and Agent A will instead transport in a given direction if it is stacked under Agent B. Furthermore, in both scenarios, Agent A's movement should be random. Given the uniqueness of modulating pattern implementation, it helps to go through an example.

In section 3.3 a simulation was presented with 4 agents: A Rat Agent with a Male and Female shape, a Box Agent, A Pizza Agent, and a Background Agent (see Figure 18). Furthermore, the Random Movement Pattern was added our Rat Agent. At this point of the simulation, both shapes of the Rat Agent randomly move around the level depicted in Figure 19, and are blocked by the Pizza Agent.

Now let us say that we want the Rat Agent to Transport the Pizza Agent (in addition to being blocked by it) as it moves. We accomplish this by launching the Simulation Creation Toolkit and clicking on the Collision interacticon in Figure 22. This brings up the following window.

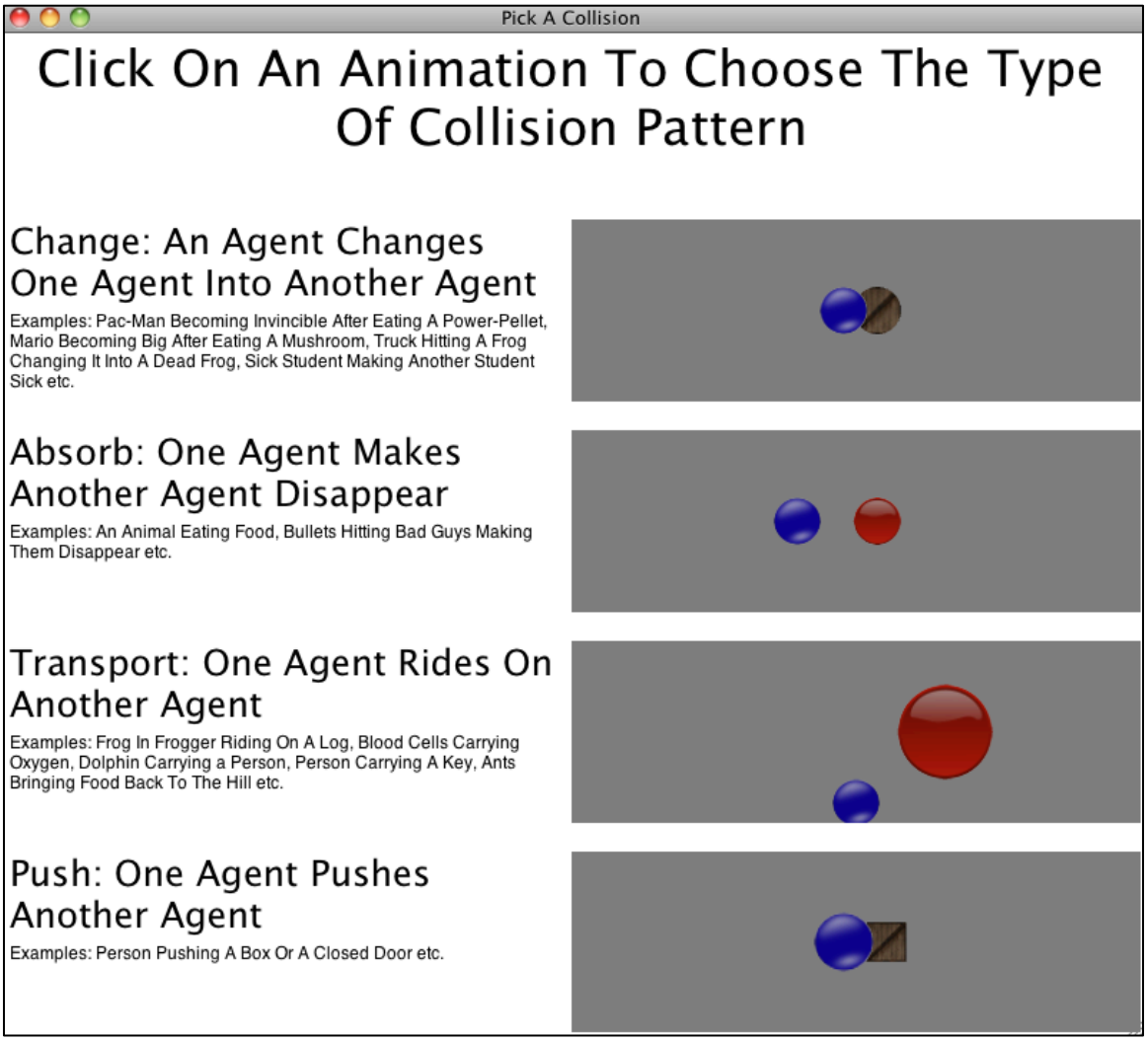


Figure 45: The Collision Picker Window

Figure 45 is the Collision Picker Window. Similar to the Movement Picker Window (Figure 23), the left side of the Collision Picker Window has descriptions of each pattern and the right side contains the corresponding interacticon for that pattern. Since we want the Transport Pattern, we click on the Transport interacticon. This brings up the following Transport Pattern Window (see Appendix B.6 for Transport Pattern Window and interacticon description).

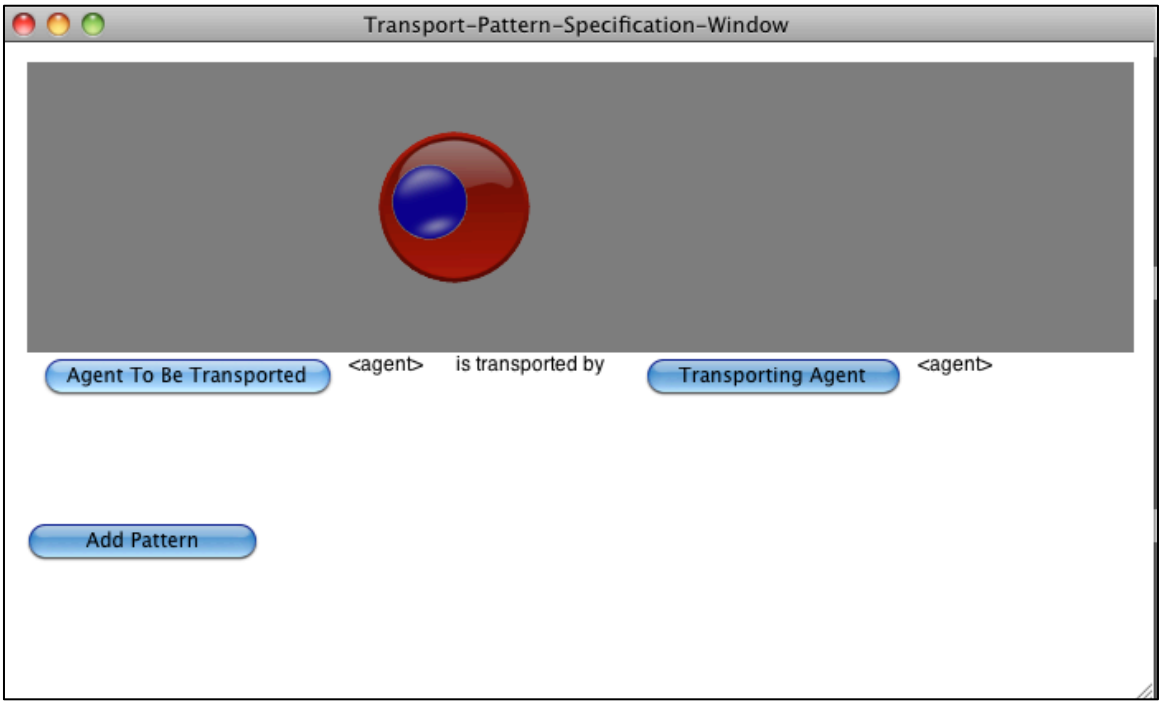


Figure 46: Transport Pattern Window

We want the Rat Agent to Transport the Pizza Agent, so we select the Pizza Agent for "Agent To Be Transported" and Rat Agent for "Transporting Agent" in Figure 46. Making these selections alters the Transport Pattern Window to look as follows.

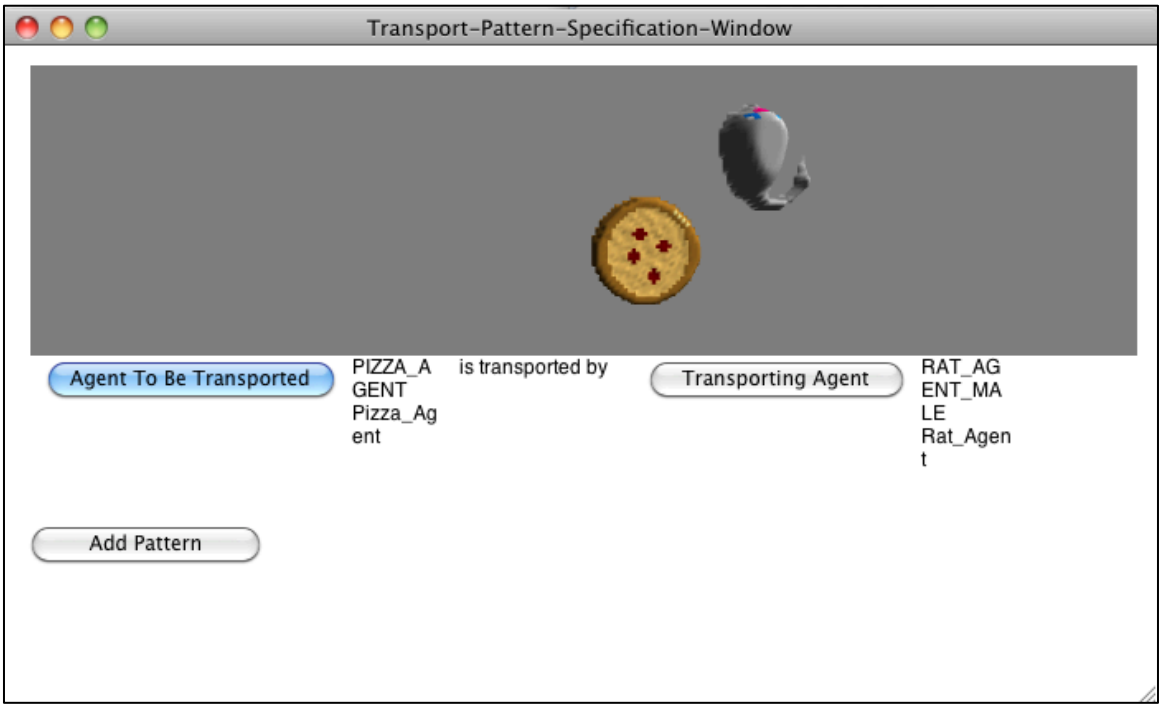


Figure 47: Transport Pattern Window With Pizza Agent Selected As the Agent To Be Transported And The Rat Agent Selected As The Transporting Agent

We now click on the "Add Pattern" button at the bottom left of the window in Figure 47. This will add the Transport Pattern Specification Box to the Simulation Construction Kit Window (originally depicted in Figure 28).

Figure 48: Simulation Construction Kit Window With Transport Pattern Specification

Notice the Transport Pattern Specification is added directly after the Random Movement Pattern Specification in the Simulation Construction Kit Window. One may also notice that the Tag Values have changed; the Tag Values changing in this section have no significance as the screenshot pictures in these examples were cobbled together from a few different simulation runs.

The Simulation Creation Toolkit does two things when implementing a modulating pattern in AgentCubes. The first is a "Placeholder" method, with no rules, is added; this placeholder method can be thought of as the pattern method for the modulating pattern. Though it contains no rules, the comment of this Placeholder method contains all the pattern specification information necessary to implement the modulating pattern (as will be shown later). The second thing is that any move rules that exist in the agent are altered and new rules are added to accommodate the modulating pattern case.

Let us first look at how the Random Movement Pattern rules, added in section 3.3, are changed. Figure 44 depicts the final rules of the Movement-Random-Movement-Tag_1 method. Since the introduction of the Transport Pattern changes all the rules in the same basic way, we will just look at how the first rule changes. In Figure 44, the first rule of the Random Movement Pattern method looks as follows.

Figure 49: First Rule Of The Random Movement Pattern Method Before The Transport Pattern Is Added

This rule is changed into the following two rules: One where the agent (in this case the Rat Agent) moves if it is not stacked under a Pizza Agent and one where the agent transports if it is stacked under a Pizza Agent.

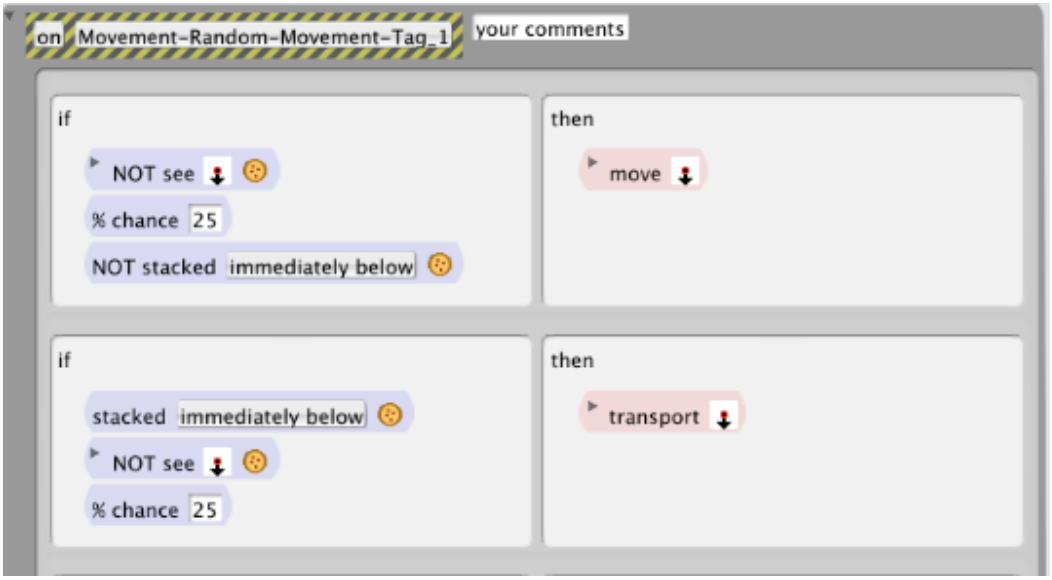


Figure 50: Random Movement Pattern Method After The Transport Pattern Is Added

The first thing to note is our original rule in Figure 49 is altered by the addition of a **not stacked condition**. The first rule now states that if the Rat Agent does not see a Pizza Agent in the downward direction, and the Rat Agent is not stacked immediately below the Pizza Agent, there is a 25% chance the rat will move in the downward direction. Additionally, another rule is added directly below wherein the conditions are copied from the first rule, with the exception that the **not stacked condition** is changed to a **stacked condition**. Furthermore, the action for this rule uses a **transport action** instead of a **move action**. Therefore, this second rule states that if the Rat Agent is stacked directly under a Pizza Agent, and there is no Pizza Agent in the downward direction, there is a 25% chance that the Rat Agent will Transport in the downward direction. Note that any conditions in the move rule that do not pertain to being stacked under a Pizza Agent, are automatically copied into the transport rule to ensure consistency between the conditions under which the agent transports and the conditions under

which the agent moves. In general, the Transport Pattern alters any rule with a **move action** by doing the following:

1) Adding a **not stacked** condition, specifying the Transported Agent's shape, into the conditions of the rule containing the **move action**

2) Adding an additional rule after this move rule with identical conditions and actions except for a **stacked condition** with the Transported Agent is used in place of the **not stacked condition,** and any **move actions** are replaced by **transport actions** in the same direction.

Table 5: How The Transport Pattern Modifies A Move Rule

This strategy enables any number of agents to be transported by a given agent; for each Transport Pattern, any move rule is modified with a **not stacked** condition and an additional rule is added for that particular Transport Pattern. At this point, the complete "Movement_Random_Movement-Tag_1" method contains eight rules; four that take care of the move condition and four that take care of the transport condition. Furthermore, if we delete the Transport Pattern, the rules will revert back to the four rules Random Movement Pattern rules depicted in Figure 44.

The Placeholder method for the Transport Pattern looks as follows.

Figure 51: Transport Pattern Placeholder Method

The Placeholder method is an empty method, but the comment of the method contains all the information as to how the pattern is specified. This comment is the string in Figure 51 that starts with "RAT_AGENT_MALE-Rat_Agent…". The Placeholder method is necessary so that if a user implements any subsequent move patterns, the Simulation Creation Toolkit knows to automatically make the additional alterations to the move pattern such that the agent transports, instead of moves, if the given Transported Agent is stacked on top of it. In short, the Simulation Creation Toolkit, after implementing the move pattern, parses out the information of the Transport Pattern Placeholder method comment and adds all the specified transport rules automatically.

At this point when we run the simulation, the Rat Agents move randomly; if we place a Pizza Agent on one of the Rat Agents, the Rat Agent transports the Pizza Agent randomly. The Rat Agents are blocked in movement by Pizza Agents but can still jump on the Box Agent. Now let us add the Push Pattern to our simulation.

3.4.1 Adding A Push Pattern To The Simulation

The other modulating pattern in the Simulation Creation Toolkit is the Push Pattern. This pattern enables an agent to push another agent in a given direction. We will now add a Push Pattern to our simulation to see how the Simulation Creation Toolkit combines both the already implemented Random Movement Pattern and modulating Transport Pattern with an additional modulating pattern. As we go through this example, note that the user creates very complicated code with just a few quick pattern implementations and specifications using the Simulation Creation Toolkit.

For this example, let us say we want the Rat Agent of any shape to push the Box Agent. Furthermore, let us say that we want this push to be blocked by both Rat Agent Shapes (so a Rat Agent cannot push a box on top of another Rat Agent). We would go through and add the Push Pattern similar to how we added other patterns in the previous sections (see section 3.3 and 3.4). After the pattern addition, we would make the following specifications to the pattern.

Figure 52: Push Pattern Specification

Upon making these specifications, the rule depicted in Figure 50 is updated to the following (we will again look at just one rule).

Figure 53: The Move Down Rule In The Random Movement Method With The Push And Transport Modulating Patterns Applied

The first rule now says that the Rat Agent only moves down (with a 25% chance) if it does not see a Box Agent or a Pizza Agent in the downward direction, and if it is not stacked immediately below a Pizza Agent. An additional rule, added by the modulating Push Pattern, appears second in Figure 53. This added rule states that if the Rat Agent sees a Box Agent in the downward direction, and it is not stacked immediately below a Pizza Agent, there is a 25% chance the Rat Agent will call the method "Push-Down-Tag_5" on itself. The "Push-Down-Tag_5" method looks as follows.

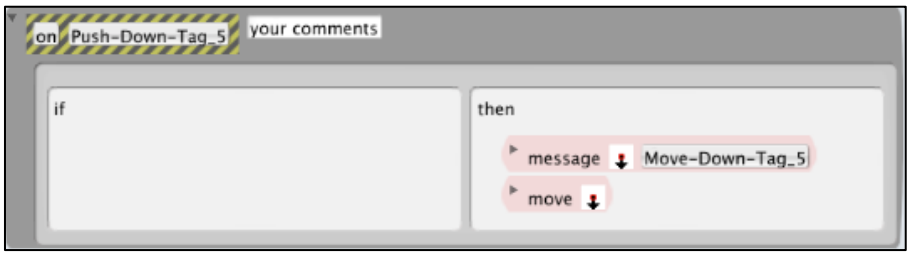


Figure 54: Push-Down-Tag_5 Method

This method sends a message to the agent in the downward direction (remember that the agent in the downward direction has to be a Box Agent based on the second rule in Figure 53) to run its Move-Down-Tag_5 method; This is an example of two different pattern methods in two different agents having the same Tag Value. Then the Rat Agent itself moves down. At this point we can think of the Rat Agent being on top of the Box Agent.

The Box Agent's Move-Down-Tag_5 method looks as follows.

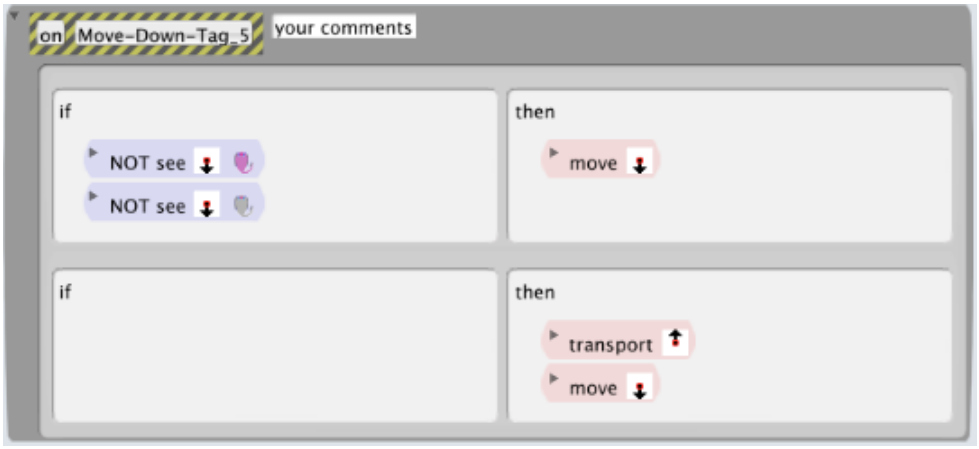


Figure 55: The Box Agent's Move-Down-Tag_5 Method

The method in Figure 55 states that if the Box Agent does not see a Rat Agent in the downward direction, the Box Agent moves in that direction (remember that the Rat Agents block the push as depicted in Figure 52). Otherwise, if the Push Pattern is blocked by one of the Rat Agent Shapes, the Box Agent Transports the Rat Agent that is now on top of it back to the Rat

Agent's original location and then moves itself back to its original location. Note that this all happens before the user sees it in the world, so from the user's perspective, the Box and the Rat Agent do not move; since this all happens in the Rat Agent's execution cycle, no other agents in the world should move or be changed.

The third rule in Figure 53 is the updated Transport rule from Figure 50; this rule is now altered to make sure there is no Box Agent in the direction it happens to be transporting, otherwise, it has to push the Box Agent as its transporting the Pizza Agent. The final rule in Figure 53 covers this case; this rule, added by the Push Pattern, states that if the Rat Agent is under a Pizza Agent and there is a Box in the direction it wants to move, then the Rat Agent calls the Push-Transport-Down-Tag_5 method on itself. This method looks as follows.

Figure 56: The Push-Transport-Down-Tag_5 Method

This method is very similar to the Push-Down-Tag_5 method depicted in Figure 54 except that instead of moving down, the Rat Agent now transports in the downward direction. However the same Move-Down-Tag_5 message is passed to the Box Agent. The Move-Down-Tag_5 method works regardless of if the Pusher Agent is transporting something or if it is not. Keep in mind that these are how the rules are modified for one direction; thus, the final Random Movement Method will have these modifications for

every direction. Furthermore, we have a Push method for every direction in the Rat Agent as well as a Push-Transport method for every direction in the Rat Agent. Finally we have a method for each of the four directions the Box Agent could get pushed.

The reason we outline these two modulating patterns is to show the challenges that emerge as patterns combine. Specifically, unlike other patterns, the modulating patterns are non-trivial because they do not just add their own rules to a brand new method. Instead, modulating patterns update the rules of currently and subsequently implemented patterns as well as update these pattern methods with additional rules. Furthermore these illustrative examples provide a good starting point for fully understanding what each specification choice in the Simulation Creation Toolkit creates in terms of AgentCubes code. This, in turn, will enable an in-depth discussion of each pattern including specification and implementation.

3.5 Description Of Specification Choices In The Simulation Creation Toolkit

We will now look at each possible specification choice for any pattern and what these choices add or remove from the pattern's AgentCubes code. We do this before we look at each pattern because many of these specification choices are repeated throughout all the patterns. Recall that pattern specifications are important because they act as pattern parameters, and thus, can be thought of as simulation parameters as well. Keep in mind that not every specification is in every pattern, just the ones that are relevant to that pattern and the ones that user might commonly use for a particular pattern. Furthermore, often a pattern will employ combinations of specifications when implemented.

3.5.1 The Agent Specification

For most patterns, the only specification a user is allowed to do within a particular pattern window is to specify which agent or agents shapes will be involved in the pattern (see Figure 27 for an example). However, the user is allowed to change this in a particular pattern specification box, which appears in the Simulation Construction Kit Window. An example of this can be found in Figure 30 and Figure 31 in section 3.3.1. Also note that in section 3.3.1 we outline that any subsequent specification that refers to the Agent Shape is automatically changed when the Agent Shape enacting the pattern is changed. An example of this is also depicted in Figure 31.

3.5.2 Speed (i.e. Once Every) Specification

The Once-Every or Speed Specification enables a pattern to happen at a given time increment. An example of this can be found in label **2** of Figure 29. The user can change this by clicking on the textbox in the specification and updating it to the time she or he wants. The Once-Every specification determines the timer method that a particular pattern is called from. These methods are talked about in-depth above (see section 3.3.3)

3.5.3 For All Depictions Of Specification

The For All Depictions Of Specification checkbox enables a user to say whether a pattern is applied to a particular shape of an agent or all the shapes of that particular agent. In certain cases wherein two agents enact a pattern together, there is a separate For All Depictions Of checkbox that corresponds to each agent. Enabling a pattern to be applied for all depictions of a given agent can mean a few things in terms of implementation. First of all, if an agent is currently checking in its pattern method rules to see if it is a specific shape using the **see condition**, this condition is deleted.

Otherwise, if an Agent is looking for a specific shape of another Agent being involved in the pattern using the **see condition** with a direction specified, this **see condition** is deleted and replaced with a **see-a condition**. There are other special cases; for example, if any shape of an agent is to be transported, then the Transporting Agent will look for a **stacked-a condition** that refers to any shape of a given Transported Agent as opposed to a **stacked condition** that refers a one shape of the Transported Agent.

3.5.4 Blocking Agents Specification

For the Push Pattern and all movement patterns, with the exception of the Tracking Pattern, the user is allowed to specify 10 Blocking Agent Shapes. A depiction of what this specification looks like can be found in label **5** of Figure 29. As mentioned in section 3.3.1, the only difference between picking a shape in the Blocking Agents Specification and picking a shape in the Agent Specification is that the Blocking Agents Specification allows you to select a Question Mark Shape (which is also the default shape for the 10 Blocking Agent Shapes); this is to enable a user to select an incorrect Blocking Agent, and later, correct this mistake by selecting the Question Mark Shape. The Question Mark Shape, in turn, is ignored by the Simulation Creation toolkit when implementing the Blocking Agents in AgentCubes.

Any Blocking Agents specified are implemented in AgentCubes by the addition of a **not see condition** in the direction a given agent intends to move to or push another agent to. Figures 44 and 55 depict this for the Random Movement Pattern and the Push Pattern respectively.

3.5.5 Agent Is Allowed To Move On Specification

The Agent Is Allowed To Move On Specification is the opposite of the Blocking Agents Specification. It is only used in the Tracking Pattern; for the

purposes of Tracking it makes more sense to specify what agents the Tracking Agent can move on. These agents in turn are the agents that diffuse the scent of the Agent being chased (using rules similar to the ones outlined in Figure 10).

3.5.6 Percent Chance Specification

The Percent Chance specification enables a pattern method rule to be triggered at a given percent. For example, if we had one Agent Generate another agent we might want this to happen with a given percent chance (as is the case when Tunnel Agents Generate Truck Agents in Frogger). This defaults to 100% in the Percent Chance Specification. If the user changes this number to 50%, a **percent chance condition** is placed in each pattern method rule with the value of 50%.

3.5.7 Direction Or Any Direction

For patterns that occur in a certain direction (and many times any direction) the user can use a direction box to specify one direction or click a checkbox to specify any direction. This specification looks as follows.

Figure 57: The Direction Specification

Note that a specific direction can be specified in Figure 57 by using the Direction Pop Up Menu. This menu looks as follows.

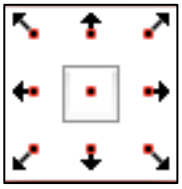


Figure 58: The Direction Pop Up Menu

The middle choice refers to the current location of the agent itself. The user can also select the Any Direction checkbox to have this pattern occur in any direction. If the user specifies a direction and also specifies any direction, then the any direction version of the rule is the one implemented. This specification could be used with the **see condition** or a **move action** for example.

Examples of patterns that use this specification are the Generate Pattern, the Absorb Pattern, the Change Pattern, and the Directional Movement Pattern (though Directional Movement Pattern does not have an Any Direction choice). This will be covered when we discuss these patterns in-depth later; however for now we will briefly review what the Direction Specification, in general, might determine. The Direction Specification determines parts of conditions and actions. For example, if an agent was absorbing or changing another agent/shape it might use a **see condition** in a given direction to see if that agent/shape exists in that direction. Similarly, if a user picks the Any Direction choice for the Absorb or Change Pattern, eight rules are added to the agent with the **see condition** covering all eight different directions an agent can see. Similarly, the direction in the Generate Pattern specifies the direction used in the **new action**. Furthermore, if All Directions (it is called All Directions as opposed to Any Direction because the Generated Agent is created in all the directions around the Generating Agent) is specified for the Generate Pattern, then there are eight **new actions** put into the same Generate Pattern method rule; one for each direction.

3.5.8 See Specification

Related to the Direction Specification (3.5.7) is the See Specification. The See Specification executes a pattern only if it sees a given shape or agent in a given direction or any direction around it. Therefore, it usually contains a direction specification in addition to a shape specification. It sometimes also contains an option to see any shape of a given agent in a given direction. A depiction of the See Specification (from the Generate Pattern Specification Box) is as follows.

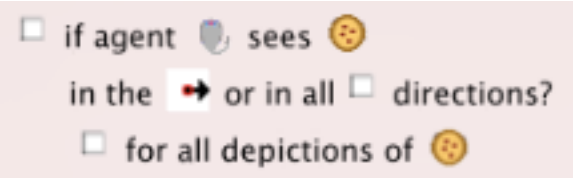


Figure 59: The See Specification

If the top checkbox in Figure 59 is checked, then the Generator Agent (in this case the Male Rat) only Generates if it sees a Pizza Shape in the Right Direction. Notice that this would not trigger for all Pizza Shapes unless the For All Depictions Of the Pizza Agent checkbox is selected. This is implemented in AgentCubes by placing a **see condition**, for a specific agent shape, or a **see-a condition**, for every shape of a given agent, in front of any Generate Pattern method rules.

3.5.9 Stacked Specification

The Stacked Specification allows a pattern to be executed only if a given Agent/Shape happens to be stacked in a certain formation relative to another Agent/Shape. For example, in the Generate Pattern Specification, the Stacked Specification looks as follows.

Figure 60: The Stacked Specification

In this case one can invoke the **stacked condition** by hitting the checkbox on the left of Figure 60. The next shape, to the right of the checkbox shown in Figure 60, is automatically determined by the user selecting the Rat Agent as the Generating Agent/Shape; therefore this Stacked Specification states that the Generating Agent/Shape only Generates when it is stacked in some formation above another Agent/Shape. The next selectable item in the Stacked Specification is the grey box with the string "immediately above" written inside. This box is a drop down menu that enables the user to state where the Generating Agent is stacked relative to another agent. When clicked, this drop down menu looks as follows.

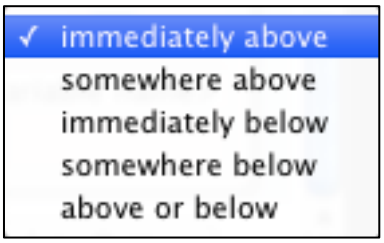


Figure 61: Stacked Specification Drop Down Menu

Using the drop down menu in Figure 61, a user can pick any stacked formation appropriate. The final choice a user can make in Figure 60 is the agent/shape the Generating Agent/Shape is stacked relative to. This is the regular shape pop-up menu. The Stacked Specification is implemented in AgentCubes by using the **stacked condition** or the **stacked-a condition** for a specific shape or agent respectively.

3.5.10 Keep Track In Variable Specification

For certain patterns that are discrete in nature, a user might want to keep track of how many times this pattern gets executed in a variable. For example, a user might want to see how many times a Fox has eaten a Rabbit in a Predator Prey Simulation, or a user might want to see how many times an alien was destroyed in a game like Space Invaders (i.e. this variable might be used to calculate a players score). The Keep Track In Variable Specification looks as follows.

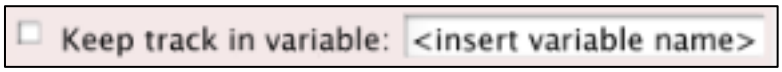


Figure 62: Keep Track Of Specification

To keep track of an event the user selects the Keep Track In Variable checkbox and inserts the variable name in the textbox wherein the text "<insert variable name>" currently resides in Figure 62. This specification is implemented in AgentCubes by using the **set action** in the rules of a pattern method with the user specified variable incremented by one each time the pattern occurs.

There are two types of variables a user can specify in AgentCubes: global or local. If the user prepends a "@" character in front of the variable name they type in, then the variable is global and can be accessed by all the instances of the agents in the world. This is useful in cases where the user wants to see how many times this pattern happened in total. Otherwise, if the user does not prepend the "@" character, the variable is local to that specific instance of the agent on the world enacting the pattern. This is useful if the user wants to know how many times a specific instance of an agent in the world has enacted a given pattern. In this version of the Simulation Creation Toolkit, the "@" character is always prepended to the variable name

as it was thought that certain students might get stuck figuring out how to create global variables. Subsequent versions of the Simulation Creation Toolkit should make this option explicit in the specification box itself.

3.5.11 If Key Is Hit Specification

The If Key Is Hit Specification is used when a pattern occurs upon a user hitting a specific keyboard key. This specification is used in the Generate Pattern (i.e. Poacher Agents shoot Bullet Agents at Fox Agents when the spacebar is hit) and the Keyboard Control Movement Pattern. The specification looks as follows for the Generate Pattern case.

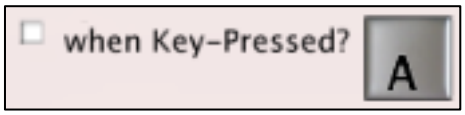


Figure 63: If Key Is Hit Specification

The checkbox in Figure 63 allows the user to specify that this pattern occurs on a keyboard hit. The "A" in Figure 63 can be changed by a user clicking on the key and hitting any key they would instead want to use for the generate. The implementation of this pattern in AgentCubes uses a **key condition**.

Now that we have reviewed all the specifications that can be used in different patterns, including the conditions and actions that correspond to these specifications, we will take an in-depth look at the patterns themselves.

3.6 Description Of Patterns In The Simulation Creation Toolkit

This section will review the 10 patterns of the Simulation Creation Toolkit with more detail. Each pattern section has 5 parts to it: General Description, Examples, Specification Choices, Implementation in

AgentCubes, and Possible Implications of Implementation. Furthermore, each pattern section contains sample AgentCubes pattern method code that corresponds to a given pattern specification; because of space constraints, the simplest version of the pattern specification will be shown (i.e. the examples that do not produce a huge amount of AgentCubes code). In this section, the quoted words refer pattern specification choices (the bold words still refer to conditions and actions).

3.6.1 The Change Pattern

The Change Pattern Window and interacticon description can be found in Appendix B.4.

General Description: Let us say we have three Agents each with a specific shape: Agent A, Shape A; Agent B, Shape B; and Agent C, Shape C. If we want Agent A, Shape A to Change Agent B, Shape B into Agent C, Shape C we use the Change Pattern specifying Agent A, Shape A as the Changer Agent; Agent B, Shape B as the Agent To Be Changed; and Agent C, Shape C as the Shape To Be Changed Into. Generally, this change occurs when Agent A is next to Agent B in some direction. It should be noted that we can also change Agent B into another shape belonging to Agent B; i.e. we may not want to change Agent B into a completely different agent, but rather, just change it to another Agent B Shape.

Examples: The Change Pattern is useful in simulations wherein an agent changes state or changes into a completely different agent. For example, in an Epidemiology Simulation, if we have a Sick Person Agent next to a Healthy Person Agent, we might use the Change Pattern to change that

Healthy Person Agent into a Sick Person Agent with a given percent chance (this percent chance would probably be based on the susceptibility for that particular Person Agent). In this case "Healthy" and "Sick" would probably be different shapes of the same "Person" Agent. Similarly, in a Predator/Prey simulation we might have a Fox Agent change a Rabbit Agent into a Dead Rabbit Agent or Shape.

Specification Choices: The following picture depicts specification choices the user can select from for the Change Pattern; the numbers in the figure correspond to the descriptions that follow.

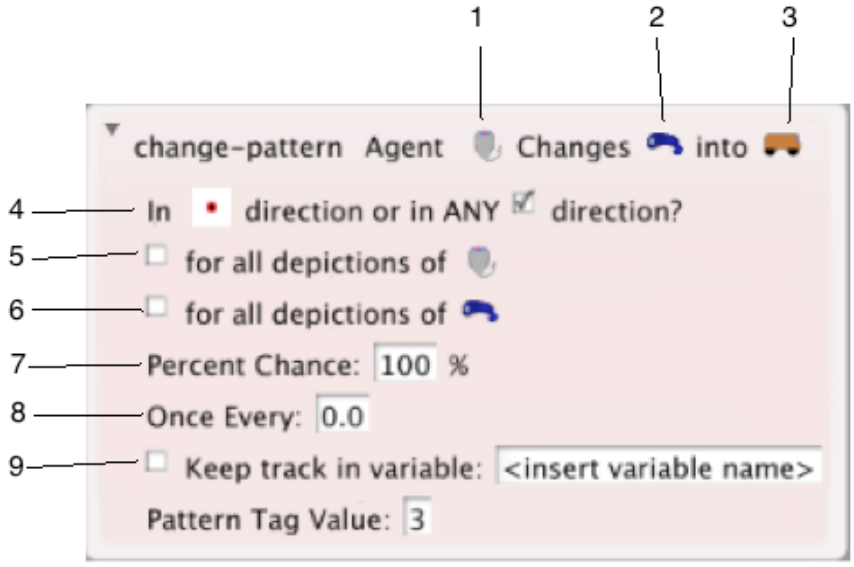


Figure 64: The Change Pattern Specifications

- 1. Changer Agent Shape: The shape that does the changing.
- 2. Shape Of Agent To Be Changed: The shape that is changed.
- 3. Change Into Agent Shape: The agent that the "Shape Of Agent To Be Changed" is changed into.

- 4. Direction Or Any Direction: The "Changer Agent Shape" can change the "Shape Of Agent To Be Changed" when next to this agent shape in a specific direction or any direction.
- 5. For All Depictions Of Changer Agent: If this is selected, every shape of the "Changer Agent" changes the "Agent Shape To Be Changed" into the "Change Into Agent Shape".
- 6. For All Depictions Agent To Be Changed: If this is selected, every shape of the "Agent To Be Changed" is changed by the "Changer Agent" into the "Change Into Agent Shape".
- 7. Percent Chance: The pattern has a given percent chance of occurring. Thus, if not set at 100% and the rest of the conditions are met, this pattern still may or may not occur.
- 8. Once Every: This pattern happens once every some odd seconds. This defaults to instantaneous time (once every 0.0 seconds).
- 10. Keep Track In Variable: The user can specify a variable to keep track of the number of times this Change Pattern occurs.

Implementation In AgentCubes: For the Change Pattern, all the rules are put in the "Changer Agent" behaviors; i.e. the selection of this agent determines wherein the behaviors for the Change Pattern are added. If all the conditions are met, the "Changer Agent" uses the **erase action** in a particular direction and a **new action** in that same direction with the "Change Into Agent Shape" being specified as the new shape to be created. The following picture depicts an example Change Pattern method implemented in AgentCubes with the accompanying pattern specification.

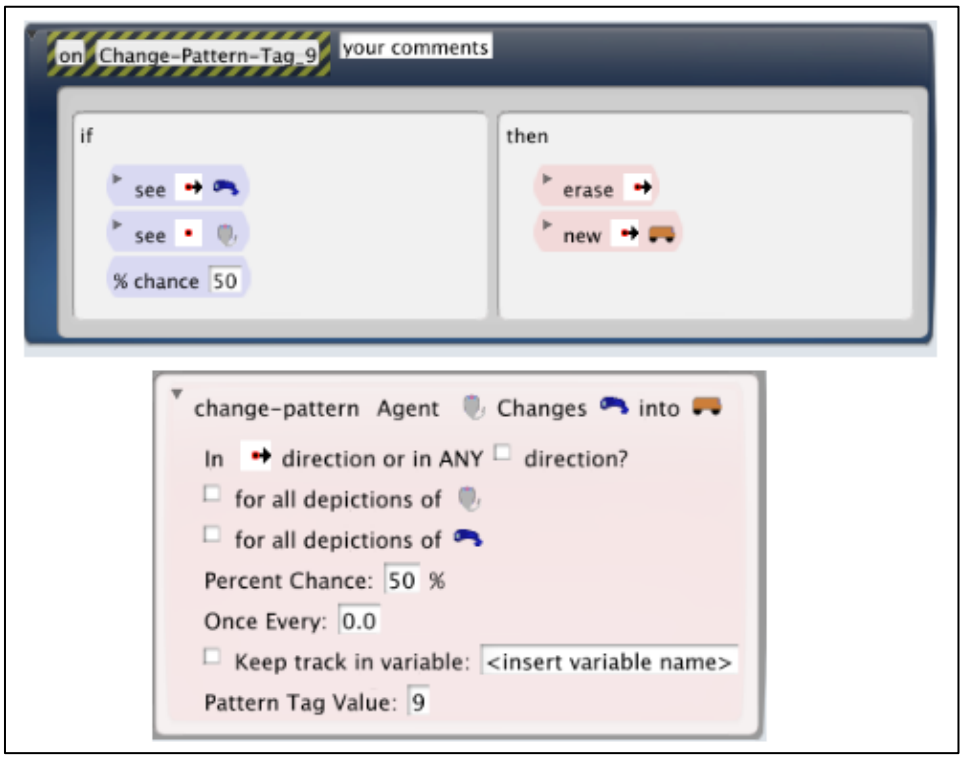


Figure 65: Example Change Pattern Method (Top) With Corresponding Specification (Bottom)

In Figure 65, we see the "Direction Specification" determining which direction the change occurs. In this case, it is to the right. Therefore, the "Changer Agent" uses a **see condition** in the right direction to determine if the "Shape Of The Agent To Be Changed" exists in that location. If it does, then the "Changer Agent" uses an **erase action** in the right direction and a **new action** in the right direction specifying the "Changed Into Agent Shape."

In Figure 65, if we selected "For All Depictions Of The Changer Agent" checkbox in the Change Specification, then the "Changer Agent" would not check if it is a particular shape using the **see condition**, as it presently does.

Similarly, if we selected "For All Depictions Of The Agent To Be Changed" checkbox in the Change Specification, then the "Changer Agent"

would use a **see-a condition** in the right direction to determine if any shape of the "Agent To Be Changed" was in that direction.

If the "Any Direction" specification is selected, 8 rules, one for each direction, would be added to the "Changer Agent" pattern method in Figure 65. In these rules, the "Changer Agent" would check to see if the "Agent To Be Changed" or the "Shape Of Agent To Be Changed**"** existed in each direction (depending on whether the "For All Depictions Of Agent To Be Changed" checkbox is selected). Each rule would be similar to the rule depicted in Figure 65, but with a different direction. The reason that we must add 8 rules instead of using the **next-to condition** has to do with the fact that we must know the specific direction we want the change to occur in order to erase and place a new agent in that direction.

The rest of the specifications are pretty straight-forward. The "Percent-Chance" specification adds a **percent-chance condition** to the rule with the percentage specified by the user. In Figure 65 this is set to 50%. The "Once Every" specification dictates which timer method will call the Change Pattern method (see section 3.3.3). The "Keep Track In Variable" specification would add a **set action** to the rule in Figure 65 that would increment some global variable.

Possible Implications Of Implementation: In AgentCubes the **change action** does not actually change an agent from one agent to another. It merely changes that agent's shape; the behaviors of that agent post change are the behaviors of the original agent (even if the shape it is changed to does not belong to that original agent). This often leads to mass confusion in classrooms as an agent will be a certain shape, but will have behaviors that are not included in that shape's agent's rules.

In contrast, the Change Pattern in the Simulation Creation Toolkit actually swaps one agent for another agent, behaviors and all. However, the mechanism for accomplishing this involves erasing the original agent and creating a new agent. This means that if the original agent had any local variables associated with it, they are now lost. However, given my experience in classrooms, this is more in line with what students expect to happen when they change one agent into another agent.

The specification should allow an agent to change if it is stacked relative to another agent. This is an oversight and should be fixed in subsequent versions of the Simulation Creation Toolkit. In this case, the "Changer Agent" could not see or pass a message to the "Agent To Be Changed" because they would be in a stacked formation. However, the "Agent To Be Changed" could check if it is stacked in some formation relative to the "Changer Agent", and if it is, delete itself and create a new "Agent To Be Changed Into." Surprisingly, in AgentCubes, this works even if the "Agent To Be Changed" deletes itself before creating a new agent (as long as both actions are in the same rule).

The "Any Direction" specification defaults to the 8 directions around the "Changer Agent". It should be noted that sometimes students might want to check just the 4 non-diagonal directions (or maybe just the 4 diagonal directions). This is not accommodated in the current implementation of the Change Pattern. Ideally, future versions of the Simulation Creation Toolkit should allow for this. A user could still accomplish using the Simulation Creation Toolkit in a tedious manner by implementing this pattern four times, once for each direction.

3.6.2 The Absorb Pattern

The Absorb Pattern Window and interacticon description can be found in Appendix B.5.

General Description: The Absorb Pattern is a collision pattern wherein one agent makes another agent disappear in a given direction, or if it is stacked in a given formation. For example, if we have two agents next to each other, Agent A and Agent B, Agent A absorbs Agent B by erasing Agent B.

Examples: The Absorb Pattern is used in many simulation contexts. For example, a Fox Agent might absorb a Rabbit Agent in a Predator/Prey simulation to indicate that the Fox Agent has eaten the Rabbit Agent.

Specification Choices: The following picture with numbered labels depicts the Absorb Pattern specification choices the user can select from; the descriptions that follow refer to these numbers.

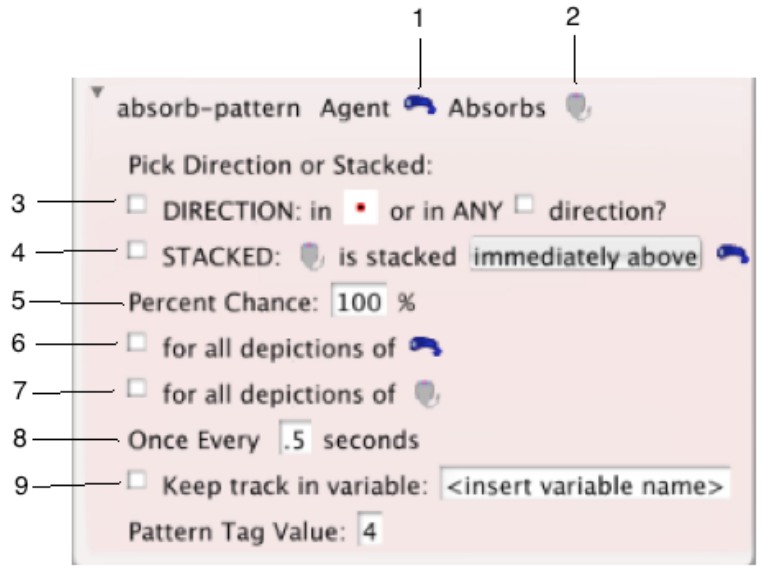


Figure 66: The Absorb Pattern Specifications

- 1. Absorber Agent Shape: The shape that does the absorbing.
- 2. Shape Of Agent To Be Absorbed: The shape that is absorbed.
- 3. Direction Or Any Direction: The "Absorber Agent Shape" can absorb the "Shape Of Agent To Be Absorbed" when next to this agent shape in a specific direction or any direction.
- 4. Stacked: Instead of specifying a direction, the user can instead specify that the "Shape Of Agent To Be Absorbed" gets absorbed if it is stacked in some formation below or on top of the "Absorber Agent Shape."
- 5. Percent Chance: This pattern has a given percent chance of occurring. Thus, if the rest of the conditions are met, this pattern may or may not occur.
- 6. For All Depictions Of Absorber Agent: If this is selected, every shape of the "Absorber Agent" erases the "Agent Shape To Be Absorbed."

- 7. For All Depictions Agent To Be Absorbed: If this is selected, every shape of the "Agent To Be Absorbed" is absorbed by the "Absorber Agent."
- 8. Once Every: This pattern happens once every some odd seconds.
- 9. Keep Track In Variable: The user can specify a variable to keep track of the number of times this pattern occurs.

Implementation In AgentCubes: The Absorb Pattern has two different basic implementations in terms of actions. If the "Absorber Agent" is specified to absorb in a given direction, then the "Absorber Agent" uses a **message action** to message the "Agent To Be Absorbed"; this in turn calls a method on the "Agent To Be Absorbed" wherein it deletes itself by using the **erase action**. This means that the "Absorber Agent" and the "Agent To Be Absorbed" both contain pattern methods with the same Tag Value.

If, on the other hand, the absorption of the "Agent To Be Absorbed" occurs when it is stacked in some formation relative to the "Absorber Agent", all the rules are placed in the "Agent To Be Absorbed" itself; the "Agent To Be Absorbed" checks if it is stacked in the user-specified formation, and if it is, deletes itself by using the **erase action**. The following depicts an implementation of the stacked case with its corresponding specification; this code would be found in the "Agent To Be Absorbed" behaviors.

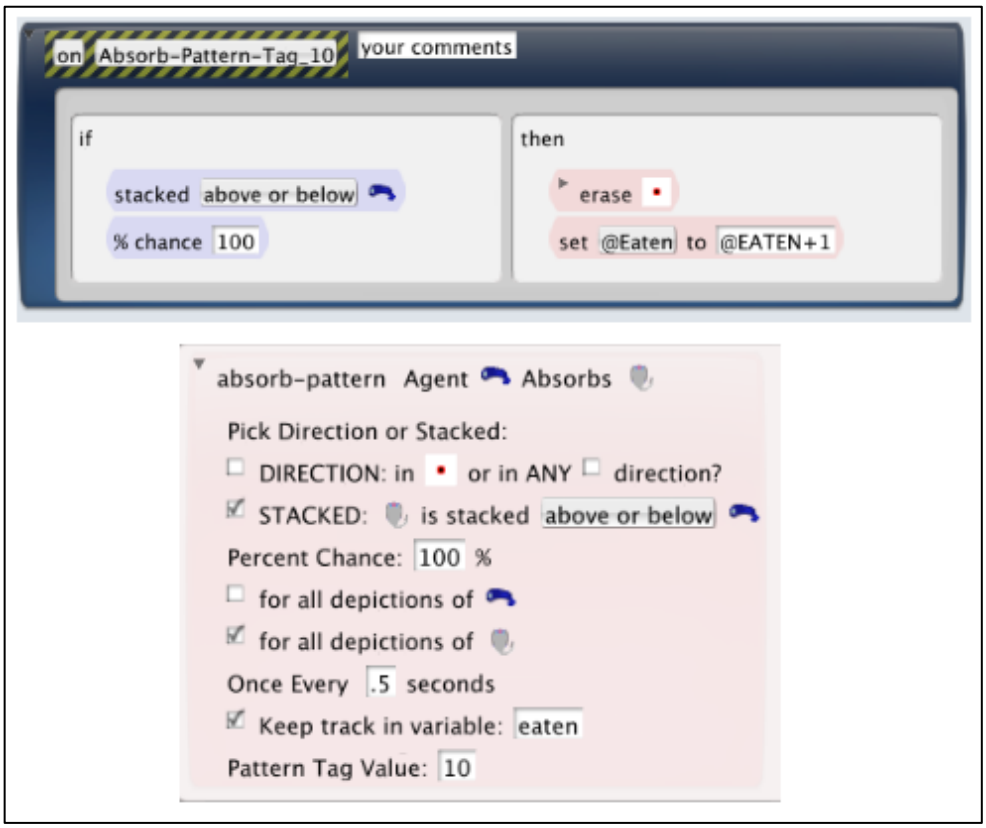


Figure 67: The Absorb Pattern Method For The Stacked Case (Top) With Corresponding Specification (Bottom)

The "Stacked" specification case is depicted in Figure 67. There are two conditions in this method's rule. The first denotes that the "Agent To Be Absorbed" must be stacked above or below the "Absorber Agent." In the bottom Absorb Pattern Specification picture we see why this is the case. The "Stacked" specification checkbox is selected, and the above or below formation is selected from the "Stacked" specification dropdown menu,

The second condition in Figure 67 denotes that there is a 100% chance this pattern will occur. The "Percent Chance" specification defaults to 100% and was not changed; therefore, this pattern will always happen.

Also note in Figure 67, the "For All Depictions Of The Agent To Be Absorbed" specification is selected. If this were not selected, the "Agent To Be Absorbed" would check to make sure it is the "Agent To Be Absorbed Shape"

by using a **see condition** on itself in its pattern method rule. Similarly, if the "For All Depictions of The Absorber Agent" specification checkbox were selected, the pattern method depicted in Figure 67 would employ a **stacked-a condition** as opposed to a **stacked condition**.

The "Once Every" specification dictates which timer method will call the Absorb Pattern method (see section 3.3.3). In the stacked case it resides in the "Agent To Be Absorbed."

The "Keep Track In Variable" specification checkbox in Figure 67 is selected with a variable named "eaten." Therefore, the pattern method also contains a **set action** wherein this variable is incremented.

As mentioned above, the "Direction" specification case sends a message to the "Absorber Agent" to erase itself. The following depiction is the same specification as above, but with the "Direction" specification selected to the right, instead of the "Stacked" specification.

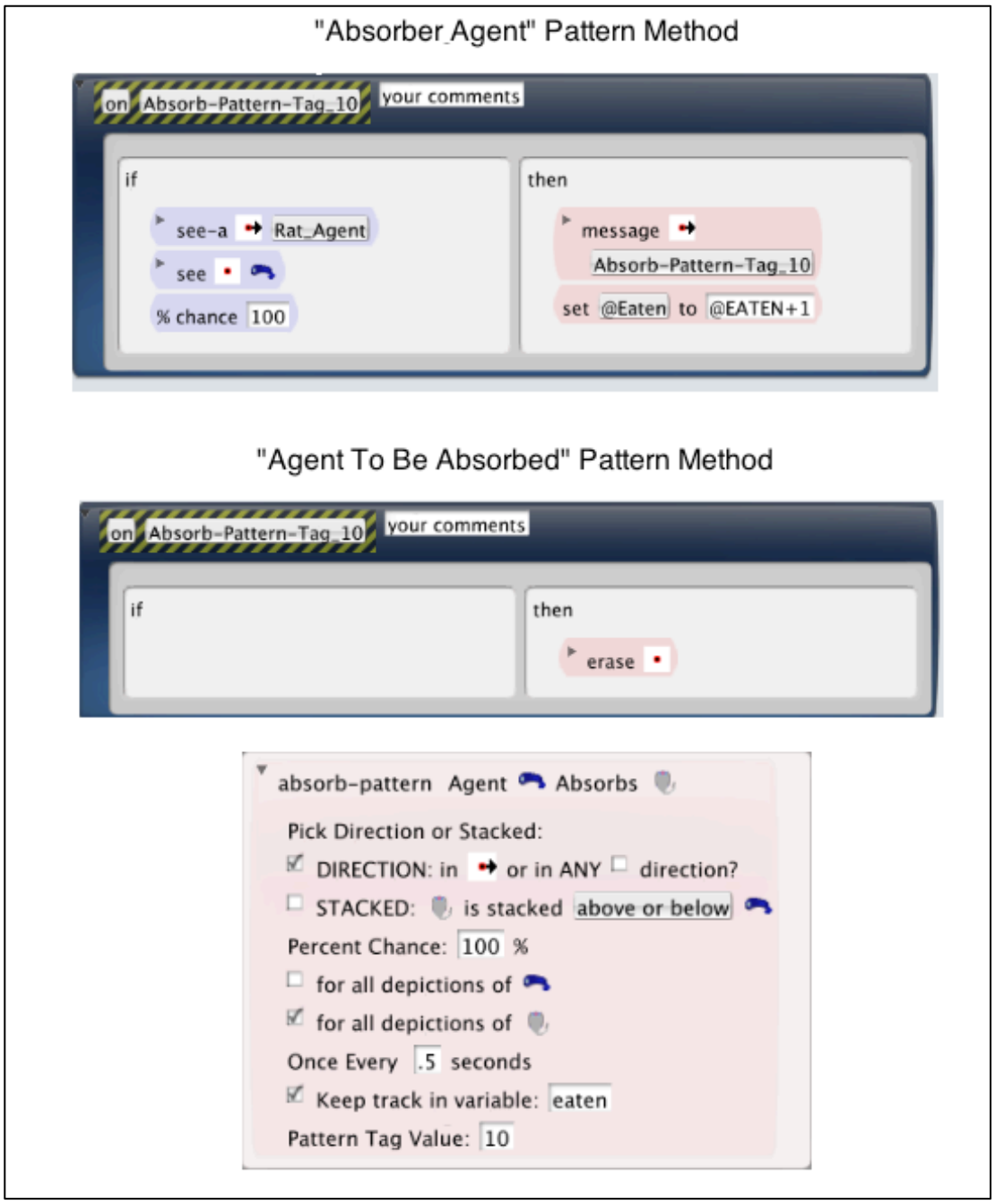


Figure 68: The Absorb Pattern Methods (Top, Middle) For The Direction Case With Corresponding Specification (Bottom)

In Figure 68 we see that now all the conditions are put in the "Absorber Agent" pattern method. The "Agent To Be Absorbed" pattern method only contains an **erase action** that erases itself. The only other difference is the "Absorber Agent" now checks its own shape using a **see**

condition because the "For All Depictions Of The Absorber Agent" specification is not selected. The timer method call, corresponding to the "Once Every" specification, will now also reside in the "Absorber Agent's" behaviors.

One major difference in the "Direction" specification is that if the "Any Direction" checkbox were to be selected, similar to the Change Pattern, the "Absorber Agent" pattern method would suddenly contain eight rules to check for the "Agent To Be Absorbed" in every direction, and if one exists, sends a **message action** to erase in that direction

.

Possible Implications Of Implementation: The Absorb Pattern has two specification choices to trigger an absorption: "Direction" and "Stacked". However, if a user tries to specify both, the system defaults to the "Direction" specification. There is no good reason that the user should not be able to specify that the absorption happens in a direction and if it is stacked. This is something that should be changed in subsequent versions of the system; however, it should be noted that a user could accomplish this by implementing this pattern twice, once with "Stacked" specified and once with "Direction" specified.

As with the Change Pattern, if the "Any Direction" specification is selected, the 8 directions around a given "Absorber Agent" are checked. This should be modified so that the users can also choose to check just 4 directions around the agent instead. Again, as with the Change Pattern, a user could get around this by implementing the pattern four times.

3.6.3 The Transport Pattern

The Transport Pattern Window and interacticon description can be found in Appendix B.6

.

General Description: The Transport Pattern occurs when one agent carries another agent. The Transport Pattern is a collision pattern because it occurs when two agents come together; however, it is also a modulating pattern because it changes every prior and subsequent implemented move pattern (see section 3.4). If Agent A Transports Agent B, then every time Agent A moves it must check if it is under Agent B, if it is not it, can move as usual. However, if it is, Agent A must instead transport Agent B in that direction. Transport means that if the Transported Agent is stacked immediately above the Transporting Agent, the Transported Agent moves with the Transporting Agent.

Examples: In simulations, examples of Transport would include a Red Blood Cell Agent transporting an Iron Agent or a Bird Agent transporting a Worm Agent.

Specification Choices: The following picture depicts the Transport Pattern specifications the user can pick from. It should be noted that much of the specifications for modulating patterns are determined by the patterns they modify; for example, the Transport Pattern rules should closely mimic any move rules present in an agent's behaviors. An example of this can be seen in section 3.4. The specification descriptions following the depiction refer to the numbered labels.

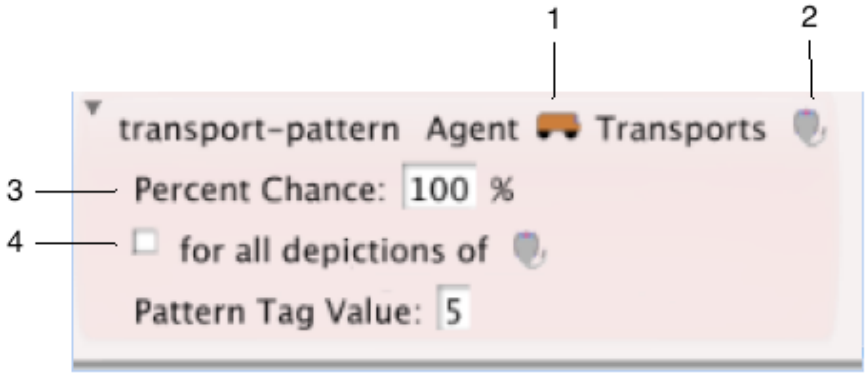


Figure 69: Transport Pattern Specification

- 1. Transporting Agent Shape: The shape that does the transporting.
- 2. Transported Agent Shape: The shape that is transported.
- 3. Percent Chance: The implementation of the Percent Chance Specification was actually taken out. It was decided that any Percent Chance that a Transport has in its condition should follow the Percent Chance of any move pattern it is modulating to avoid ambiguity. Subsequent versions of the Simulation Creation Toolkit should take out this specification option.
- 4. For All Depictions Of Transported Agent: If this is selected every shape of the "Transported Agent" is transported by the "Transporting Agent Shape."

Implementation In AgentCubes: This information is covered in-depth in section 3.4.

Possible Implications Of Implementation: The Transport Pattern makes some assumptions that might not be true. First off, it assumes that the user would only want the "Transporting Agent" to transport the "Transported Agent" if it is immediately below the "Transported Agent." The

reason for this assumption is that if the "Transporting Agent" is not immediately below the "Transported Agent" it will end up transporting agents not explicitly specified by the user to transport. However, the nature of the **transport action** in AgentCubes means that this might occur anyways.

The **transport action** transports every agent that happens to be above the agent that is transporting. Therefore, let us say we have three agents: Agent A, Agent B, and Agent C. Furthermore, let us say a user implements a pattern wherein Agent A transports Agent B. If Agent A is stacked directly below Agent B and Agent C is stacked above Agent B, Agent A will transport carrying both Agent B and Agent C with it. There is no apparent elegant solution to this problem because Agent A cannot pass Agent B a message after it moves as Agent C is on top of Agent B. Subsequent iterations of the Simulation Creation Toolkit should find a way to solve this issue.

There is no "For All Depictions Of Transporting Agent" specification. The assumption is that the "Transporting Agent" rules that contain a **move action** will specify whether or not just one shape of the "Transporting Agent" or all the shapes of the "Transporting Agent" transport the "Transported Agent"; meaning, the move rule conditions that get copied to create the transport rule will specify the shape it has to be. It could be possible that a user has a "Transporting Agent" that moves the same for all its depictions, however, just one depiction of the "Transporting Agent" is meant to transport the "Transported Agent." However, if a transport rule copies a move rule's conditions, but specifies a different shape than that move rule, it could lead to a contradiction in the transport rule rendering it un-executable. On the other hand, if the transport rule somehow had a different shape requirement

than the corresponding move rule, it could lead to unintended consequences such as an agent shape's motion being incorrect when it happened to be transporting.

This problem is in part due to the Simulation Creation Toolkit creating code automatically for the user. The Transport Pattern has to affect every implemented move pattern in the "Transporting Agent." When applying the Transport Pattern, the user does not have the ability to apply this pattern to just part of an agent's code. The user could get around this in the Simulation Creation Toolkit by splitting up shapes that transport and the shapes that do not into different agents.

3.6.4 The Push Pattern

The Push Pattern Window and interacticon description can be found in Appendix B.7.

General Description: Like the Transport Pattern, the Push pattern is not only a collision pattern wherein the pattern occurs when two agents come together, but also, a modulating pattern that depends on any movement rules that have been implemented before it. Agent A is said to Push Agent B if Agent A attempts to move into Agent B's grid space, and when Agent A moves into Agent B's grid space, Agent A supplants Agent B's original position moving Agent B one space in the direction of Agent A's movement.

Examples: Examples of Pushing include an Elephant Agent pushing a Rock Agent or a Human Agent pushing a Box Agent.

Specification Choices: The following picture depicts the Push Pattern Specification Choices; the descriptions that follow refer to the numbered labels in the following figure.

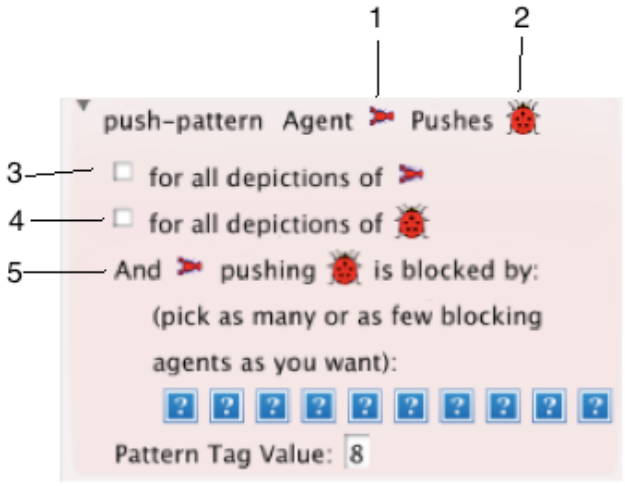


Figure 70: Push Pattern Specification

- 1. Pusher Agent Shape: The shape that does the pushing.
- 2. Pushed Agent Shape: The shape that is pushed.
- 3. For All Depictions Of Pushing Agent: If this is selected every shape of the "Pushing Agent" pushes the "Pushed Agent."
- 4. For All Depictions Of Pushed Agent: If this is selected every shape of the "Pushed Agent" is pushed by the "Pusher Agent."
- 5. Pushing Is Blocked By: These 10 boxes allow the user to define 10 different shapes in the simulation that disallow the "Pusher Agent" to push the "Pushed Agent" onto them.

Implementation In AgentCubes: An in-depth example of Implementation in AgentCubes can be found in section 3.4.1.

One thing to note, the Push Pattern is similar to the Transport Pattern in that it adds a Placeholder method with all the information pertaining to the pattern contained within the comment of that method. This enables the Simulation Creation Toolkit to modify any subsequently implemented move patterns with the rules that enable the "Pusher Agent" to push the "Pushed Agent."

Possible Implications Of Implementation: The Push Pattern only enables the user to specify up to 10 "Blocking Agents." This was done for implementation simplicity reasons; however it would make more sense for the user to add or delete the number of "Blocking Agents" depending on what the user needs for a particular simulation.

The Push Pattern assumes that the agent is only moving and pushing in 4 directions. However, it could be possible that the user wants to move in and Push in 8 different directions. This is partly due to the fact that any move pattern only allows agents to move in 4 directions. Subsequent versions of the Simulation Creation Toolkit should enable 8 directional movement in addition to the 4 direction movement currently implemented. If the movement patterns are updated to have 8 directional movement, the Push Pattern should automatically work with no modification as it is based on the implemented move rules.

The way that the Push Pattern is implemented is such that it works with all the movement patterns and the Transport modulating pattern. In the situation wherein the "Pusher Agent" is not at the top of the stack and pushes the "Pushed Agent", and that push is blocked by a "Blocking Agent", the result is not entirely correct. This is because the "Pusher Agent" will move on top of the "Pushed Agent", and the "Pushed Agent" will transport the

"Pusher Agent" back to its original grid space in the world (see section 3.4.1). However, the "Pusher Agent" will now be on top of the stack instead of the location it originally was. Though minor, it is possible that this could lead to problems in a user's simulation. It was decided that this problem was more desirable than not having the Push Pattern work with every other pattern effectively. For example, unlike other implementations of the Push Pattern, this implementation of Push works with the Transport Pattern as is shown in section 3.4.1.

3.6.5 The Random Movement Pattern

The Random Movement Pattern Window and interacticon description can be found in Appendix B.9.

General Description: The Random Movement Pattern is a movement pattern that enables an agent to move randomly in 4 directions (up, down, left, and right) at a given speed. If Agent A moves randomly once every 0.5 seconds, that means that Agent A, every 0.5 seconds, will have an equal percentage of moving any of the four above specified directions.

Examples: The Random Movement Pattern is often used in simulations where masses of Agents interact with each other, but a more realistic movement is not necessary. For example, in an Epidemiology Simulation we would often use the Random Movement Pattern for the People Agents though these agents would not necessarily be moving randomly throughout the day in real life.

Specification Choices, Implementation In AgentCubes, and Possible Implications Of Implementation: Refer to Section 3.3, which covers all of these sections in-depth for Random Movement.

3.6.6 The Tracking Pattern

The Tracking Pattern Window and interacticon description can be found in Appendix B.10.

General Description: The Tracking Pattern is used when one Agent chases another Agent around the world. Let us say we have Agent A and Agent B; Agent B diffuses a scent around the world and Agent A always moves in a direction wherein the scent of Agent B is the highest (see sections 1.4.3 and 1.4.4).

Examples: The Tracking Pattern is used in Predator/Prey simulations when a Predator Agent gets hungry and starts chasing the Prey Agent. It could also be used in simulations to recreate attraction, for example, in magnet simulations or to make agents follow a given path such as blood flowing through the body.

Specification Choices: The following picture depicts the Tracking Pattern Specification Choices; the descriptions that follow refer to the numbered labels in the following figure.

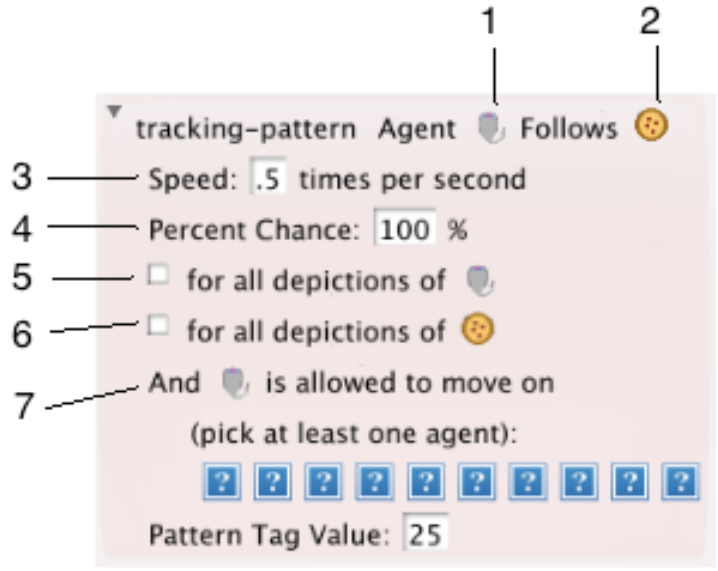


Figure 71: Tracking Pattern Specification

- 1. Tracking Agent Shape: The shape that follows the scent of the "Tracked Agent."
- 2. Tracked Agent Shape: The shape that is chased.
- 3. Speed: Specifies how fast the "Tracking Agent" moves (same as the "Once Every" specification seen in previous patterns).
- 4. Percent Chance: This does nothing and should be removed from the Tracking Pattern Specification in subsequent iterations of the Simulation Creation Toolkit.
- 5. For All Depictions Of Tracking Agent: Specifies whether every shape of the "Tracking Agent" chases the "Tracked Agent" or just the one shape chosen for the "Tracking Agent."
- 6. For All Depictions Of Tracked Agent: Specifies whether every shape of the "Tracked Agent" is chased or if it is just the one shape chosen for the "Tracked Agent."
- 7. Agent Is Allowed To Move On: This is the opposite of the "Blocking Agent" specifications. The user can pick up to 10 agent shapes that

diffuse the "Tracked Agent's" scent. Thus these agents are the ones that the "Tracking Agent" will move on when chasing the "Tracked Agent."

Implementation In AgentCubes**:** Much of the AgentCubes implementation for the Tracking Pattern can be found in section 1.4.4. Briefly we will look at how these specifications are implemented. The "Tracked Agent" sets a scent variable using the **set action** to some value (the Simulation Creation Toolkit forces this value to be set to 1000). Note that this scent value is a local variable because it only relates to one instance of the "Tracked Agent" (see Figure 9). The "Background Agents", specified by the "Agent Is Allowed To Move On" specification, sets their own "Tracked Agent" scent value based on the average of the 4 directional (up, down, left, and right) scent values around it (see Figure 10). Finally the "Tracking Agent" checks the four directions around it and chooses to move in the direction wherein the scent is the highest (see Figure 11).

The "Speed" specification determines which timer method the "Tracking Agent" method will reside (see 3.3.3). It should be noted that the speed at which the scent diffuses around the level is not determined by this speed, but rather, defaults to instantaneous time; thus, the calls to the "Tracked Agent" method and the "Background Agent" method for tracking (wherein the scent value is set) occurs from a "Timer-0-0" method.

If the "For All Depictions Of Tracking Agent" and/or "For All Depictions Of Tracked Agent" specification is not selected, then a **see condition** is inserted with the "Tracking Agent" shape and the "Tracked Agent" shape rules respectively.

Possible Implications Of Implementation: There are many things that the Simulation Creation Toolkit automatically takes care of when implementing the Tracking Pattern. For the most part, this leads to convenience, however, it also leads to a less customizable Tracking Pattern.

As with all movement patterns, the Tracking Pattern only has four direction movement. Eventually the Simulation Creation Toolkit should enable eight direction movement.

Similar to all the patterns that have blocking shapes, the Tracking Pattern only allows the user to specify up to 10 different shapes for the "Tracking Agent" to move on. Subsequent versions of the Simulation Creation Toolkit should allow the user to specify as many different agent shapes for an agent to move on. Furthermore, if a user does not specify at least one agent to allow the "Tracking Agent" to move on, the pattern does not work. This makes sense as the "Tracking Agent" should not track if it is not allowed to move on any agent; however, this is the default specification after the user generates the pattern using the Tracking Pattern Window (depicted in Appendix B.10) in the Simulation Creation Toolkit. It might be better to have this specification actually appear in the Tracking Pattern Window such that when the pattern is initially created, it has a chance of working.

A bug in the system exists if the user specifies the Tracked agent as an agent the Tracking Agent can move on. In this case the scent value of the Tracked Agent will constantly reset to zero as it will average the four values around itself for its own scent. The Simulation creation Toolkit should alter the Allowed to Move On specification to not allow users to select the Tracked Agent. Furthermore, the Tracking Agent will naturally move onto the Tracked agent when it reaches that agent anyways (as it has the highest scent value in the level).

When the Tracking Pattern is combined with the Push Pattern, it will not actually work unless the "Pushed Agent" diffuses the scent of the "Tracked Agent." This is due to the fact that the "Tracking Agent" will never try to move onto the "Pushed Agent's" grid space because it will have a scent value of 0. Therefore, the user must specify that the "Pushed Agent" is an agent that the "Tracking Agent" can move onto even though the "Tracking Agent" will never actually move on top of the "Pushed Agent" if the pattern is implemented correctly. This could possibly confuse the user.

Finally, the Tracking Pattern diffuses the scent among Background Agents with an equation that averages the four spaces around it. It is possible to have the scent diffuse wherein a certain direction is weighted higher than the other directions, or instead of an average, some other coefficient could be used to diffuse the scent. The Simulation Creation Toolkit does not allow for either of these scenarios. It should be noted that if the coefficient is high enough (>.50), then the scent diffusion gradient may not become a descending gradient or may grow as you get further from the "Tracked Agent"; thus, the "Tracking Agent's" movement will not necessarily go towards a "Tracked Agent." Furthermore, in all my experiences in using Tracking in the classroom, I have never personally seen a student try to weight a given direction differently than the other directions (though this could have interesting results).

3.6.7 The Keyboard Control Movement Pattern

The Keyboard Control Movement Pattern Window and interacticon description can be found in Appendix B.11.

General Description: The Keyboard Controlled Movement Pattern enables a user to specify the movement in the world of a given agent in four directions based on four different keyboard button presses. The four keys default to the arrow keys (as this is what most users would use for movement).

Examples: The Keyboard Control Movement Pattern is usually used in games as opposed to simulations. However, it could be used to anytime a user needs to control the position of a given agent while the simulation is running. For example, in a Predator/Prey simulation a user might make the Poacher Agent's movements controlled by keyboard placing these agents in different positions. Ostensibly, it could be possibly that a user programs switches based on where an agent is located to enable different parameter values in a simulation.

Specification Choices: The following picture depicts the Keyboard Control Pattern Specification Choices; the descriptions that follow refer to the numbered labels in the following figure.

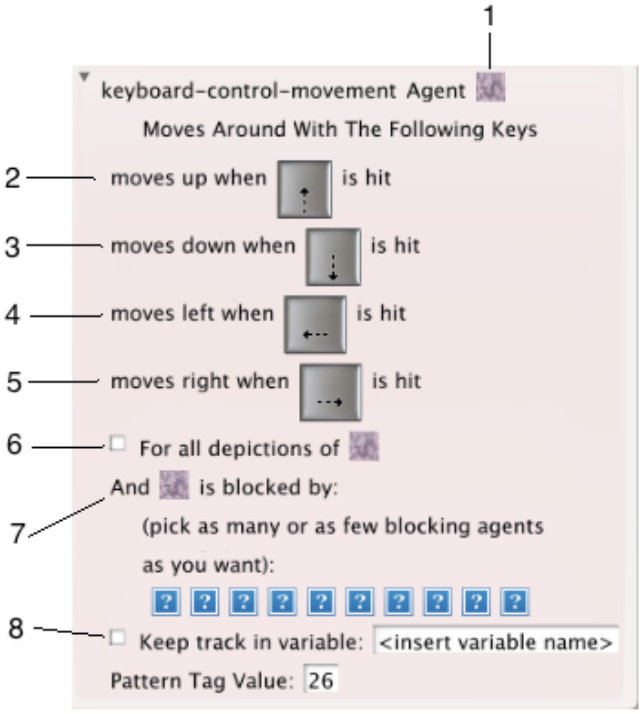


Figure 72: Keyboard Control Movement Pattern Specification

- 1. Keyboard Controlled Agent Shape: Specifies the shape that will be keyboard controlled.
- 2,3,4,5. Directional Movement Keys: This specifies the keys that will be used to move the agent up, down, left, and right respectively.
- 6. For All Depictions Of Keyboard Controlled Agent: Any shape belonging to the "Keyboard Controlled Agent" will move in response to the keys specified in the "Directional Movement Keys".
- 7. Blocking Agents: The user can select 10 shapes that block this movement.
- 8. Keep Track Of Moves In Variable: The user can specify a variable that increments every time the user moves. This is used in games like Sokoban wherein the aim is to solve the puzzle in the least amount of movements possible.

Implementation In AgentCubes: The implementation of the Keyboard Control Movement Pattern is straightforward. The "Keyboard Controlled Agent Shape's" pattern method has four rules that involve a **key condition**, which is specified by the "Directional Movement Keys", with a **move action** for every one of the four movement directions (up, down, left, and right). The following depicts an example implementation with its corresponding specification (which is overlapped to preserve space).

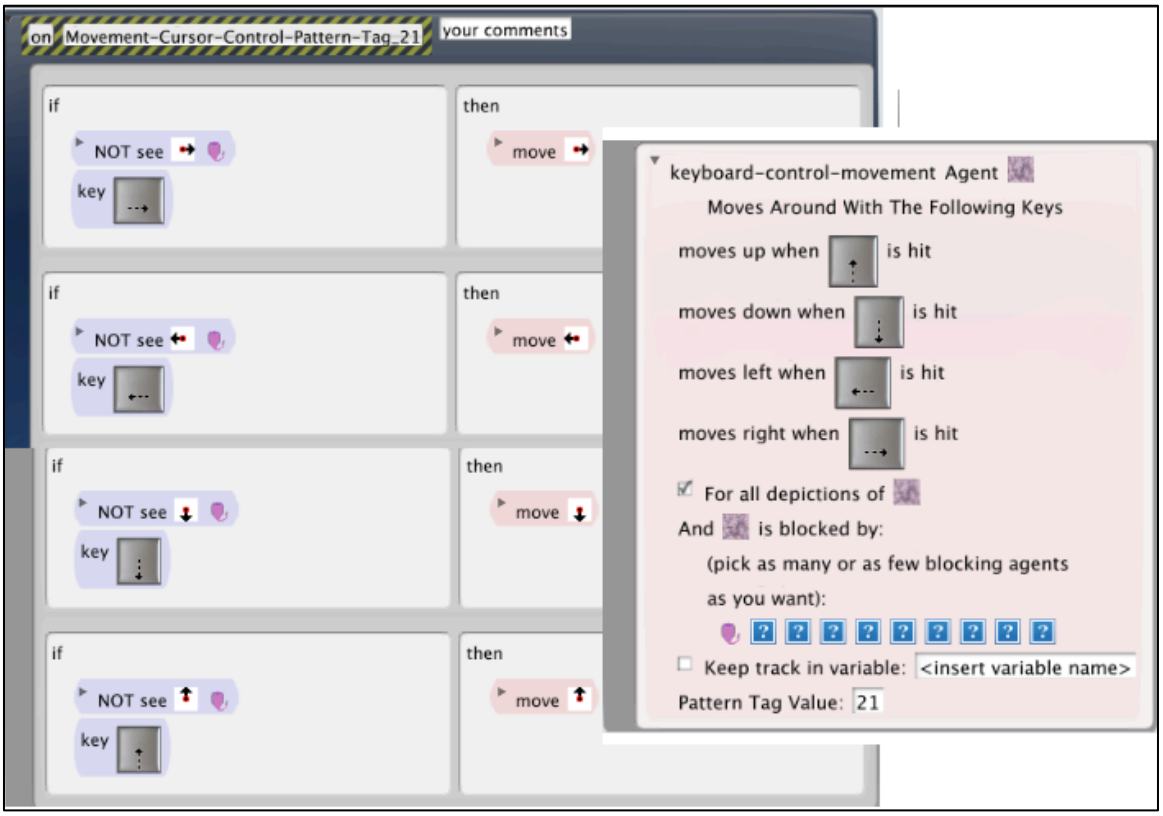


Figure 73: The Keyboard Control Pattern Method (Left) With Corresponding Specification (Right)

In Figure 73 we see that our conditions include a **key condition** and a **not see condition**; these correspond to the "Directional Movement Keys" specification and "Blocking Agents" specification respectively. Since the "For All Depictions Of Keyboard Controlled Agent" is selected in the specification,

the “Keyboard Controlled Agent” does not have another **see condition** in its rules to check for if it is a particular shape. The Keyboard Control Pattern method is called from a “Timer-0-0” method (i.e. instantaneous time); this is because the agent should react with movement immediately after the user hits a key.

Possible Implications Of Implementation: They Keyboard Control Pattern Specification is too restrictive in that users should be able to select a direction, using a “Direction Specification”, in addition to a corresponding keyboard key to trigger that direction. A better version of the pattern specification would enable users to specify as many or as few direction-key combinations as necessary.

3.6.8 The Directional Movement Pattern

The Directional Movement Pattern Window and interacticon description can be found in Appendix B.12.

General Description: The Directional Movement Pattern enables an agent to move in a given direction with a given speed. For example, let us say a simulation contains one agent: Agent A. The Directional Movement Pattern can be used to have Agent A move to the right once every .5 seconds.

Examples: Whenever an agent moves with a constant velocity (same speed and direction) then the Directional Movement Pattern can be

employed. For example, if a Poacher Agent Generates Bullet Agents, the Bullet Agent might move across the world at a given constant velocity.

Specification Choices: The following picture depicts the Directional Movement Pattern Specification choices; the descriptions that follow refer to the numbered labels in the following figure.

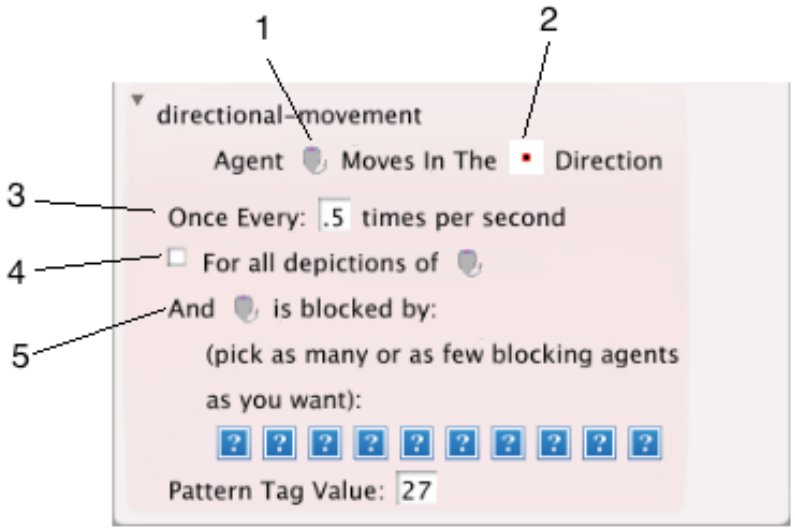


Figure 74: The Directional Movement Specification

- 1. Directional Movement Agent Shape: Specifies the shape that the Directional Movement Pattern will be applied to.
- 2. Direction Specification: Direction pop-up menu that allows the user to define the direction the "Directional Movement Agent Shape" moves.
- 3. Once Every: Defines the speed of the directional movement.
- 4. For All Depictions Of: If selected, the Directional Movement Pattern is applied to every shape of the "Directional Movement Agent."
- 5. Is Blocked By: The user can define 10 "Blocking Agent Shapes" that stop the directional movement.

Implementation In AgentCubes: The Directional Movement Pattern Method is called from a timer method based on the "Once Every" specification (see section 3.3.3). The following figure depicts an example implementation with its corresponding specification.

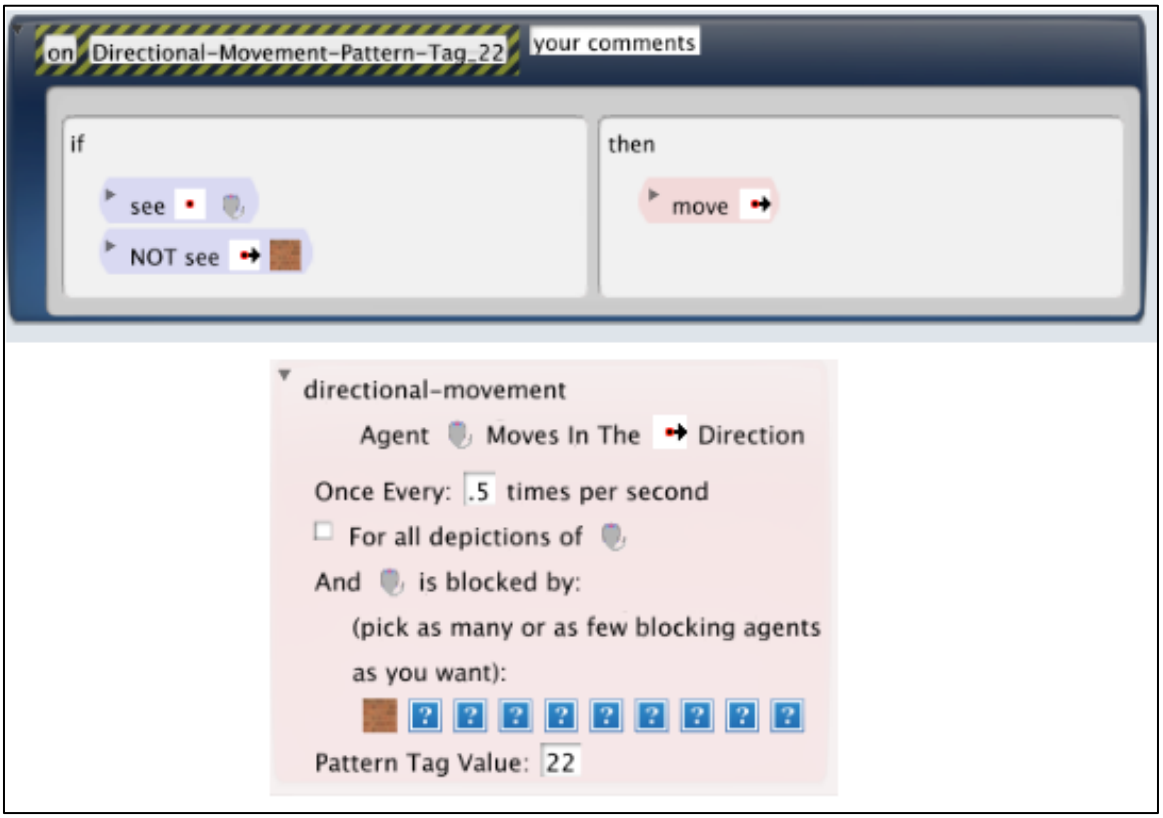


Figure 75: Directional Movement Pattern Method Implementation (Top) With Corresponding Specification (Bottom)

Figure 75 depicts directional movement to the right. The pattern method itself has one rule containing a **move action** with the direction determined by the "Direction Specification." In Figure 75 the "For All Depictions Of" checkbox is not selected, and thus, the rule has a **see condition** with the "Directional Movement Agent Shape." Finally, Figure 75 has one "Blocking Agent Shape" specified, so the pattern method rule has a

not see condition with that shape and with the direction defined as the same direction of movement (right).

Possible Implications Of Implementation: The Directional Movement Pattern is very simple; there are no real unique implications of implementation. As mentioned in previous patterns with "Blocking Agent Shapes", subsequent versions of the Simulation Creation Toolkit should allow for as many "Blocking Agent Shapes" as possible.

3.6.9 The Generate Pattern

The Generate Pattern Window and interacticon description can be found in Appendix B.13.

General Description: The Generate Pattern enables one agent to generate another agent in one or any direction, when one or many conditions are met. For example, let us say we have two agents: Agent A and Agent B. Agent A is said to generate Agent B if Agent A creates an instance of Agent B in the world.

Examples: The Generate Pattern has many uses in simulations. For example, to simulate mating in a Predator/Prey simulation, a Predator Agent might generate another Predator Agent if it is next to a Predator Agent with a given percent chance (the Prey Agent might similarly do the same thing). A Poacher Agent might generate Bullet Agents to simulate shooting. A Dirt Agent might Generate Grass Agents etc.

Specification Choices: The following picture depicts the Generate Pattern Specification choices; the descriptions that follow refer to the numbered labels in the following figure.

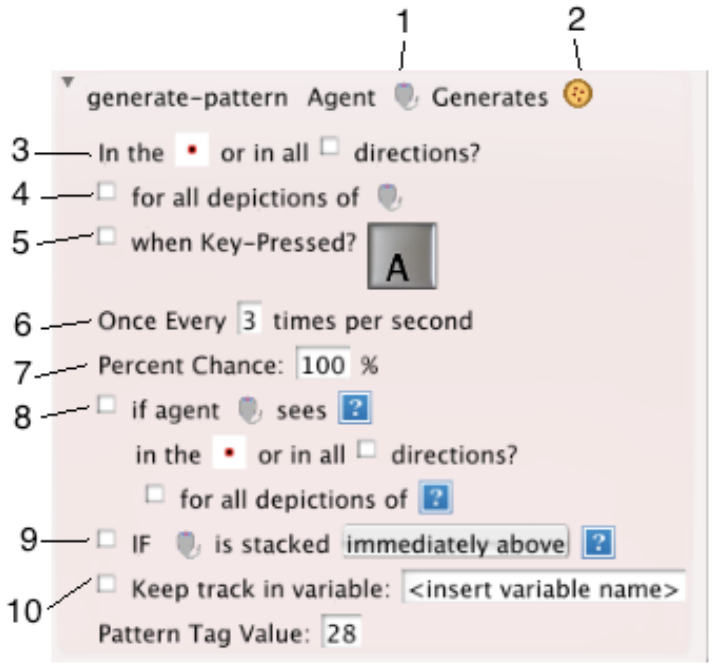


Figure 76: Generate Pattern Specification

- 1. Generator Agent Shape: Specifies the shape of the agent that does the generating.
- 2. Generated Agent Shape: Specifies the shape of the agent that is generated.
- 3. Direction Specification: Allows the user to choose which direction the generation happens in or if it happens in every direction (all eight directions).
- 4. For All Depictions Of Generator Agent: Determines if every shape of the "Generator Agent" generates the "Generated Agent Shape."
- 5. When Key Is Pressed: If selected, the generation happens when the user presses the specified key.

- 6. Once Every: The generation will happen once every specified seconds.
- 7. Percent Chance: The generation will happen with a given percent chance.
- 8. If Generator Agent Sees A Shape: If this is selected, the generation occurs if the "Generator Agent" sees the specified shape in a given direction or any direction. The second shape box automatically changes when the shape is selected.
- 9. If Generator Agent Is Stacked: If this is selected, the generation occurs if the "Generator Agent" happens to be stacked in some formation relative to the specified shape.
- 10. Keep Track Of In Variable: Enables the user to specify a variable that is incremented each time the "Generator Agent" creates another shape.

Implementation In AgentCubes: The Generate Pattern method is placed in the "Generator Agent" behaviors. The "Direction Specification" is the direction included in the **new action** of the pattern. If any direction is selected, eight **new actions** are added to the rule, one for every direction. This case is depicted in the following implementation picture with corresponding specification.

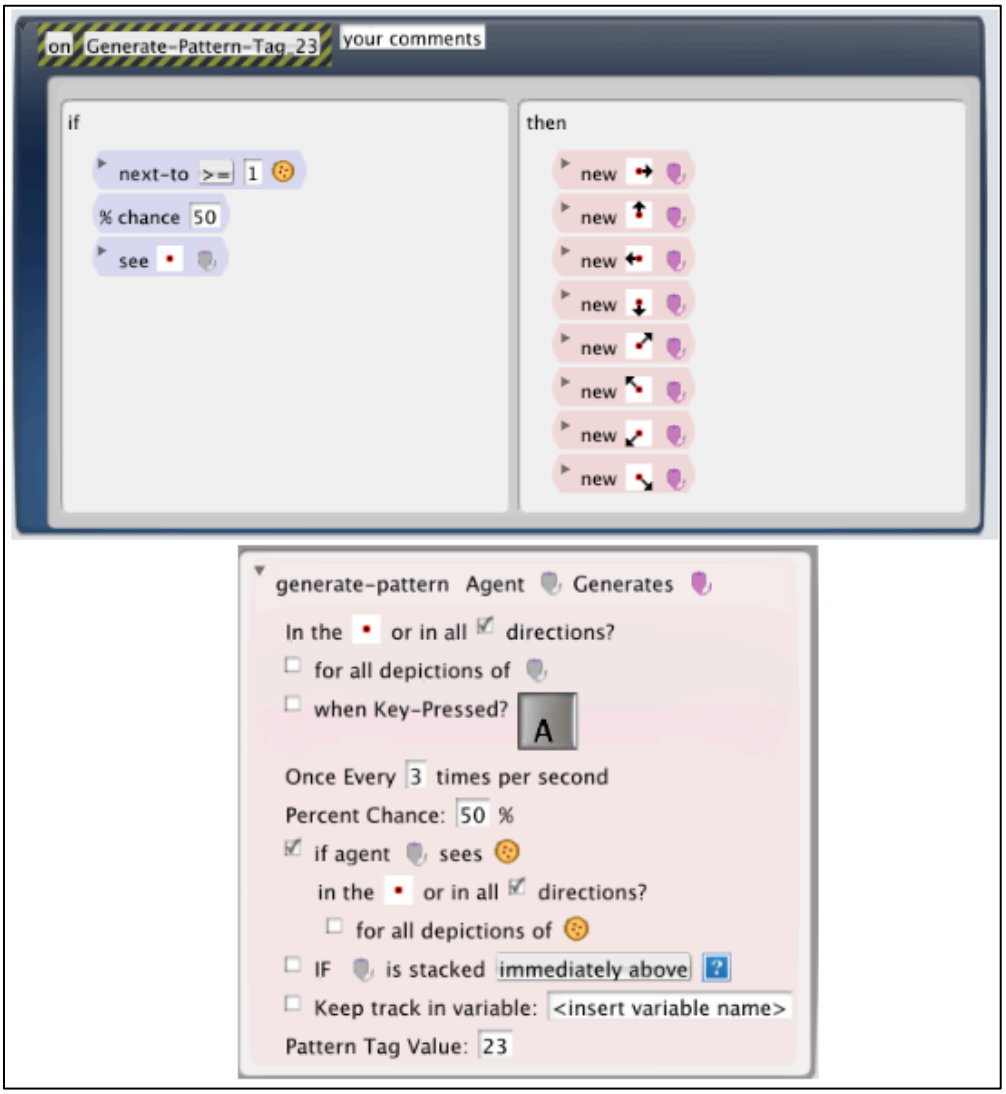


Figure 77: The Generate Pattern Method (Top) With Corresponding Specification (Bottom)

In Figure 77 the generate happens in every direction 50% of the time, if the "Generator Agent" is a given shape and happens to see a Pizza Agent in any direction. Unlike the Change and Absorb Pattern, since the "Generator Agent" does not do anything other than see the Pizza Agent, it does not have to know what direction it is located; therefore, instead of adding 8 rules with **see conditions to** check for the Pizza Agent (one for every direction), the Generate Pattern method can instead employ the simpler **next-to**

condition. In Figure 77 the "For All Depictions Of Generator Agent" is not selected, therefore the pattern method uses a **see condition** to determine if it is the "Generator Agent Shape" before creating the "Generated Agent Shape."

If the "When Key Is Pressed" specification were selected in Figure 77, a **key condition** would be added to the Generate Pattern rule with the specified key determined by the user. The "Once Every" specification determines the timer method that calls the pattern method (see 3.3.3). "If Generator Agent Is Stacked" checks if the "Generator Agent" is stacked in a certain formation to a specified shape before creating the "Generated Agent Shape"; thus this adds a **stacked condition** to the pattern rules. Finally if the user selects the "Keep Track Of In Variable" then a **set action** is added to the pattern rules incrementing the specified variable each "Generated Agent" creation.

Possible Implications Of Implementation: There are many conditions that can be combined for the Generation Pattern. Therefore, the method rules can have a huge amount of conditions. However, in practice users typically only use a subset of these condition specifications. The reason that there are so many specifications has to do with the variety of ways users tend to use this pattern. Generation can occur with a keyboard hit in a game context or once every so often if the game/simulation calls for a steady stream of agents; however, it can also happen when an agent happens to be in a certain place in the world (i.e. around certain other agents). This makes the Generation Specification somewhat complicated, but all these specification choices are necessary to enable pattern usability in a many different contexts.

Similar to the "If Generator Sees A Shape" specification, the "If Generator Agent Is Stacked" specification should ideally allow for the "Generator Agent" to be stacked not just relative to one specific shape, but possibly every shape of a given agent. The "If Generator Sees A Shape" allows for this by providing a checkbox for this specification. Subsequent versions of the Simulation Creation Toolkit should enable this; the **stacked-a condition** could be used to determine if all shapes of a given agent are stacked in a certain formation relative to the "Generator Agent." Currently, the user could get around this by implementing multiple Generate Patterns for all the possible stacked shapes.

Even with all these choices there are things that cannot be accomplished using the Simulation Creation Toolkit specifications for the Generate Pattern. For example, if a user wanted the agent to see two shapes in two different directions in order for the "Generator Agent" to create the "Generated Agent", this would not be possible given the specification choices. Similarly, if the generation were to happen if the "Generator Agent" is below one agent but above another agent in order to create, again this would not be possible given these specifications. There are other examples of specific implementations that might be impossible to implement using the Simulation Creation Toolkit (i.e. four directional generate instead of eight directions); As mentioned above, enabling users to create their own patterns via an authoring tool in the Simulation Creation Toolkit could in part solve this problem.

3.6.10 The Data Pattern

The Data and interacticon description can be found in Appendix B.14.

General Description: The Data Pattern allows a user to select an agent to be counted and an agent that accomplishes the counting as well as a global variable that keeps track of the number of agents. The number of agents on the worksheet will be placed in this specified variable.

Examples: The Data Pattern is often used in simulations to collect population information. For example, in a Predator/Prey simulation the Data Pattern can be used to track the population of Foxes and Rabbits over time. In an Epidemiology simulation the Data Pattern can be used to find out how many Healthy, Sick, and/or Dead People Agents there are at a given time.

Specification Choices: The following picture depicts the Data Pattern specification choices; the descriptions that follow refer to the numbered labels in the following figure.

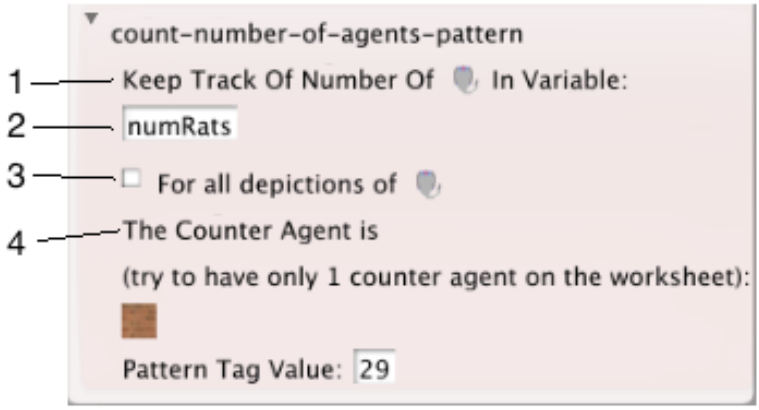


Figure 78: Data Pattern Specification

- 1. Agent Shape To Count: Specifies the shape of the agent whose population the user wants to keep track of.
- 2. Global Variable To Increment: Specifies the variable wherein the population value of the "Agent Shape To Count" is placed.

- 3. For All Depictions Of Agent Shape To Count: Specifies if every shape of the "Agent To Count" is counted.
- 4. Counter Agent Shape: The Agent that actually broadcasts a message to all the "Agent Shapes To Count" in the world to increment the "Global Variable To Increment." It is suggested that only one instance of this agent exists in the world. Often, this agent is hidden under the background agent in the world (but does not have to be).

Implementation In AgentCubes: The Data Count Pattern defaults to counting once every 2 seconds. Therefore, the call to the Data Count Pattern method is in a Timer-2-0 method. The "Counter Agent Shape" can be any agent in the world that happens to have just one instance in the world (and will not be erased etc.). Every 2 seconds the "Counter Agent Shape" sets the "Global Variable To Increment" to zero (using a **set action**) and then uses a **broadcast action** to send a message to every instance of the "Agent Shape To Count" wherein that agent increments the specified global variable yielding the population of that "Agent Shape To Count." It should be noted that if there are multiple "Counter Agent Shapes" in the world the counting should hypothetically still work, however, it will just count the agents more than it necessarily needs to (once for each "Counter Agent Shape" in existence in the world for each update cycle).

The following depicts an implementation of the Data Count Pattern methods with corresponding specification.

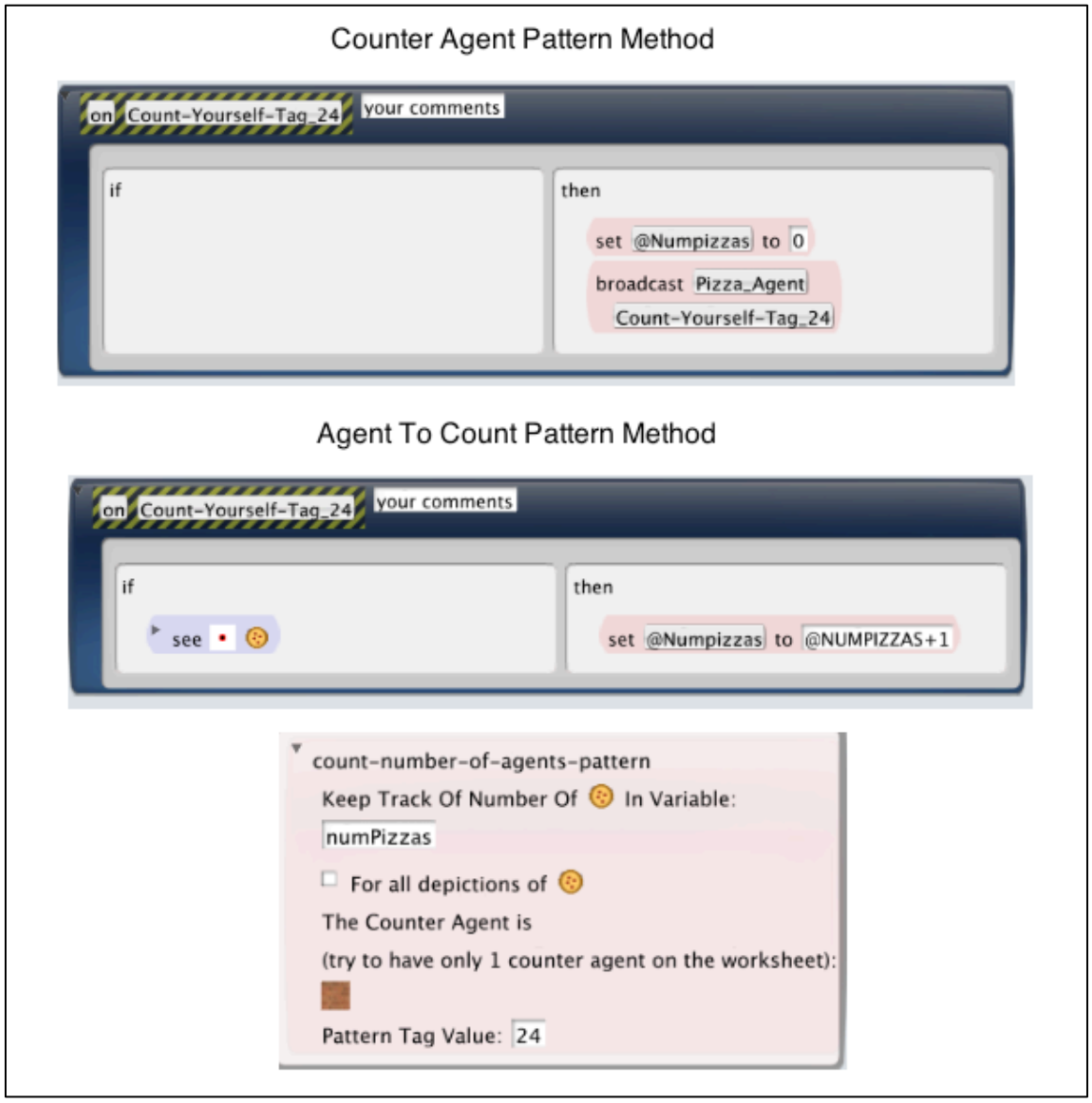


Figure 79: Data Pattern Methods (Top, Middle) With Corresponding Specification (Bottom)

Figure 79 depicts the Counter Agent using the **set action** to set a global variable, in this case "numPizzas", to zero and then using a **broadcast action** to have all the "Agent To Count" instances increment the "numPizzas" global variable. This is accomplished, in this case, by broadcasting a method named "Count-Yourself-Tag-24." The "Agent To

Count", in turn, first checks to see if it is a specific shape using a **see condition**; this is because the "For All Depictions Of Agent To Count" checkbox is not selected in the specification. Finally, it uses a **set action** to increment the variable.

Possible Implications Of Implementation: The Data Pattern is very simple. The only peculiar thing about this pattern is the request that a "Counter Agent" exists on the worksheet. This might force the user to create a new agent for the express purpose of collecting data. To avoid this issue, originally all the code for this pattern was placed in the "Agent Shape To Count." However, with many "Agent Shapes To Count" populating the worksheet, the population values flickered between 0 and the population number which might have led to confusion among students. Furthermore, this strategy seems terribly inefficient. For example, if there are 100 "Agent Shapes To Count", each of those 100 agents would set the global variable value to zero and then broadcast to all the "Agent Shapes To Count" to increment the global variable. Therefore, it was decided that having the user pick a "Counter Agent", with one instance in the world, would be a better strategy.

We have now reviewed every pattern in the Simulation Creation Toolkit. Next we will discuss how these patterns can be combined together to make sample simulations, simulation tasks that lend themselves well to this tool, as well as situations wherein one would be hard-pressed to create a simulation using this tool. Finally we will end with an analysis of the user interface.

3.7 Discussion Of The Simulation Creation Toolkit Effectiveness In Simulation Contexts

Computational Thinking Patterns started as the units of transfer between game programming and science simulations. It is possible to break down many simulations purely in terms of Computational Thinking Patterns. For the simulations where this is possible, the Simulation Creation Toolkit is effective. Computational Thinking Patterns themselves inspired some simulations that work well with this toolkit. These include Epidemiology, Predator/Prey, and Pine Beetle simulations. Working with 7th grade Life Science over the past 2 years inspired and flavored this toolkit as well. Other example simulations that work from this domain are Natural Selection and Global Warming simulations. Simulations that could possibly be created but might not be ideal include human body simulations such as Blood Flow or Cellular Respiration and simulations of Circuits. We will briefly talk about many of these simulations later in this section. It should be noted that most of these simulations fit into the Centenary Middle School Life Science Guaranteed Viable Curriculum.

However, in many cases, simulations not only employ Computational Thinking Patterns, but also, require the user to write low-level agent behaviors that are not encapsulated in the Computational Thinking Patterns. The Simulation Creation Toolkit is an initial exploration into the effectiveness of programming simulations at the Computational Thinking Pattern Level. In simulations wherein Computational Thinking Patterns are used in addition to low-level code, the Simulation Creation Toolkit effectiveness is mitigated. It is hypothetically possible for a user to add their own low-level code to programs partly created by the Simulation Creation Toolkit; the Simulation Creation Toolkit itself does not explicitly facilitate

this. For example, if a user changed a pattern manually by adding a condition, action, or rule to a pattern method, upon making any additional change to the pattern by selecting a specification, for example, the modified code would be lost and the user would have to re-add it. Instead, users would have to implement code by creating methods and adding their method calls into timer methods separate from the patterns implemented in the Simulation Creation Toolkit. Eventually this functionality should be added to the Simulation Creation Toolkit and is an integral part of further research.

Given that users cannot easily add code, it should be noted that there are certain simulations that do not lend themselves easily to creation in AgentCubes itself. For example, projectile motion is fairly difficult to conceptualize in this domain as it involves the use of recursion, and because of the grid structure, looks extremely choppy. Similarly, in AgentCubes, though possible, it might be non-ideal to create simulations of planets orbiting. The Simulation Creation Toolkit is domain oriented towards simulations based on common patterns found helpful over years of experience. Though it can save users time on particular tasks, the types of simulations that work in this domain are subset of the types of simulations that can be achieved using the finer-grained palette of conditions and actions in AgentCubes. Therefore, if something is not achievable in AgentCubes, it is pretty safe to say that it will not work in the Simulation Creation Toolkit.

The Simulation Creation Toolkit is a first step towards higher-level pattern programming of simulations. To this end it gives us an opportunity to investigate the effectiveness of pattern programming; however, it accomplishes this in a limited context with a finite amount of patterns, a finite number of specifications for these patterns, and the inability to modify implemented patterns. If programming at the Computational Thinking

Pattern level is found to be useful, these above concerns should be systematically solved.

It should also be noted that the "correctness" of a simulation is a complex topic. For what this thesis aims to accomplish in the domain of education, the idea was to create a tool that could be integrated, for example, into the 7th grade Life Science Guaranteed Viable Curriculum. To this end, the simulation creation activities undertaken by students using this tool would hopefully be ones that support multiple learning goals of this Guaranteed Viable Curriculum. However, the aim of this initial investigation is two-fold. The main goal is to create a tool that could hypothetically be used by students to create a realistic enough simulation easily that, in-part, supports these learning goals. The second goal is to see if some evidence exists to show that this exercise in simulation creation, using this tool, is useful in actually getting students to meet these learning goals. Therefore, a simulation creation exercise wherein agents move randomly might be an unrealistic representative system, however, might still effectively support the learning goals for a given topic and be effective in integrating computational thinking concepts into the classroom..

It should also be noted that though an important point of simulation creation is to construct a correct model of the real world that helps in a particular problem domain, this thesis is interested in seeing if students in classrooms can effectively use the Simulation Creation Toolkit. Theoretically, if this tool is something students can use to create a variety of simulations that tie into the Guaranteed Viable Curriculum, the correctness of a given simulation can be fine-tuned later. For example, in the Predator/Prey simulation, the percentages used for the Fox eating a Rabbit might not be

entirely correct, but at some point, these can be modified in the lesson plan to better reflect the correct behavior of these agents in the simulation.

Furthermore, given the inherent limited nature of this toolkit, in any simulation there will come a point when making it realistic, or tailoring it to a given problem, becomes too complicated or even impossible for students using the Simulation Creation Toolkit. In these cases, it would be up to the teacher to decide if the simulation they could create using the Simulation Creation Toolkit would be at a level that would benefit student-knowledge of the subject as opposed to hinder or confuse students on a particular topic.

The idea behind this is an important discussion point in the Simulation Creation Toolkit, namely, if a simulation is created that is incorrect in some aspect, either visually or behaviorally, it might do more harm to the student as it might lead to misconceptions or confusion. The Simulation Creation Toolkit offers a variety of patterns to accomplish a given simulation creation task. In some sense it is up to the teacher to ensure that the patterns used to create a given simulation teach the ideas that match the Guaranteed Viable Curriculum correctly. In some cases, it might not be beneficial to attempt a simulation creation exercise that is not conducive to this tool. However, if an authoring functionality were added to the toolkit, this problem might disappear altogether as the necessary patterns could then be created.

3.7.1 A Discussion Of Simulation Exercises Using The Simulation Creation Toolkit

To better understand the Simulation Creation Toolkit, it helps to look at a few examples of the types of simulations that can be accomplished and the types of simulations that might be troublesome using this tool.

Since the Simulation Creation Toolkit was made in the context of seventh grade Life Science at Centenary Middle School, it might help to look at a few examples from this curriculum. Some examples of things that the Simulation Creation Toolkit can accomplish include Predator/Prey simulations, Natural Selection simulations, and Epidemiology Simulations. We will describe the Predator/Prey unit in-depth, as it is the one that we used for this study, and then talk about the other two units.

Based on past AgentSheets in-class units, the Predator/Prey simulation taught usually includes the following elements. Each element is described followed by an explanation of how it might be accomplished using the Simulation Creation Toolkit.

- Predator and Prey Move Randomly: The Random Movement Pattern can be applied to both the Predator and the Prey.
- Predator/Prey becomes hungry at some point: The user can create a Hungry Shape for the Predator and the Prey Agent. A Background Agent can use the Change Pattern to change these agents into their hungry counterparts with a percent chance. Thus, as time goes on, a Predator and/or a Prey has a greater chance of becoming hungry.
- Predator tracks Prey when it is hungry: This is accomplished by using the Tracking Pattern with the Background Agent diffusing the Prey scent.
- Prey forages for plants when it is hungry: Again, this is accomplished by using the Tracking Pattern.
- Hungry Predator eats Prey, Prey disappears; hungry Prey eats Plant Agent, Plant disappears. Upon doing this, the Predator/Prey is no longer hungry: This can be accomplished

by having the Predator change the Prey into a Dead Prey Agent, and the Dead Prey Agent changing the Hungry Predator back into the default Predator Shape. Seconds later, the Dead Prey Agent can be Absorbed by the Background. Similarly, the Plant Agents can be changed into an Eaten Shape that changes the Hungry Prey into the default Prey Shape, before being absorbed by the Background etc.

- If the Predator/Prey stays hungry for too long they die: The Change Pattern can be used with a Background Agent Changing these hungry agents into dead agents with a percent chance.
- Dead Predator/Prey creates Plant Agents: Instead of making the dead agents disappear, The Background Agent can change the Dead Predator/Prey Agents into Plant Agents over time representing decomposition.
- Predator mates with Predator to create more Predators; Prey mates with Prey to create more Prey Agents: The Generate Pattern can be employed to have this mating occur with a percent chance when a Predator is next to another Predator Agent for example.

Depending on how complex and/or realistic a teacher would want to make this simulation, interactions could be added or taken out. For example, a teacher might have students calculate the trophic levels of the system based on the biomass, taken from agent population data, of the Predator, Prey, Prey, and Plant agent. Or another Predator could be added to create competition. Furthermore, by altering pattern specifications and taking

population data students can run experiments. For example, students can use the Data Pattern to count the population of agents over time. Furthermore, students can alter the chance of mating by changing the "Percent" parameter in the Generate Pattern. It should be noted that for the purposes of Seventh Grade Life Science class, I was told my Marco Cornacineachionne that a simulation similar to this would be more than enough for students at that particular level.

An example of a natural selection simulation could be as follows. If we have three agents in competition for the same food source: Agent A diffuses towards the food source, Agent B randomly moves around the world, and Agent C randomly moves but at a much faster rate as Agent A and Agent B. Furthermore, let us say that if the Agents stay hungry long enough they die; if they are not hungry and are next to an Agent that is the same they would create more of that Agent through mating. Using the Data pattern the user could see which of these movement traits are selected for. Furthermore, by using the specifications to change different parameters (i.e. speed of the agents, what they are allowed to move on, chance of mating etc.) the user could do experiments on to what extent certain traits can lead to a species being selected for. For example, if Agent C randomly moves much faster than Agent A, and/or mates with a higher percentage, at what point would Agent C become the dominant agent in the world? A simulation akin to this can be created extremely quickly using the Simulation Creation Toolkit.

Almost the exact Epidemiology Simulation described in section 1.4 can be created with the Simulation Creation Toolkit. Briefly, We could start by making every agent randomly move. Sick Agents can use the Change Pattern to change other Agents into their Sick Agent counterparts at a given percentage (based on the susceptibility of the Agent to Be Changed that the

user can set). Furthermore, the sick agents can in turn change into dead agents or be absorbed by the background with a given percent chance. This could even be a global variable that the Absorb Pattern uses in its "Percent Chance" specification for example. Again this simulation is pretty trivial to create using the Simulation Creation Toolkit.

There are many other simulation examples from the Guaranteed Viable Curriculum that may work with the Simulation Creation Toolkit. These include Global Warming Simulations, for example, patterns can access a global temperature variable, set to the amount of Carbon Agents on the worksheet in a percent chance specification, to dictate things like Plant Agent Generation.

Also a Blood Flow simulation could be created using a network of Blood Vessel Agents as diffusing agents and having Oxygenated Blood Agents/Shapes use the Tracking Pattern to move towards various organs and Un-oxygenated Blood Agents/Shapes move to the lungs. Having the speed depend on global variable the user sets in the Simulation Properties of AgentCubes could simulate the rate of the pumping heart. Finally, organs could absorb Oxygenated Blood Agents and, upon seeing an Oxygenated Blood Agent, could create an Un-oxygenated Blood agent; this could be kept in a running tally as a global variable which could in turn indicate if the organ was getting enough blood. A similar strategy might also work for a Circuits Simulation developed using the Simulation Creation Toolkit. Depending on what the teacher wants the students to learn, these simulations might help or confuse student understanding.

Examples of a simulation that relates to Life Science that the Simulation Creation Toolkit could have trouble creating are agent

interactions that are not conducive to the pattern palette. One such interaction is the synchronized movements of many agents. It should be noted that these simulations are examples that would work in AgentCubes but not the Simulation Creation Toolkit. For example, the Amylase simulation, initially discussed in section 1.4, might not be ideal for this high level tool.

In past AgentSheets experiences, this simulation was accomplished by representing starch as individual chains of Glucose Agents connected together. To this end, the Glucose Agents move with each other until part of their chain is cut by the Amylase enzyme leading to two smaller starch chains and eventually to a single Glucose Agent. To accomplish this correctly, a pattern called Link and another pattern called Cut might have to be added to the Toolkit. Currently, the only way the Simulation Creation Toolkit could create something like this would be to have a different agent for each different sized starch chain. However, even this would not be correct or even visually correct (i.e. the chains would all look the same size but would represent different sizes). Therefore, hopefully, a teacher would choose not use the Simulation Creation Toolkit for this particular class unit.

3.8 Discussion Of The Simulation Creation Toolkit User Interface

We will now briefly discuss a few issues that arose throughout the creation and testing of the Simulation Creation Toolkit. This discussion is based on the interface elements found in Appendix B. We will first discuss the Simulation Creation Toolkit windows and then discuss the interacticon animations.

3.8.1 Discussion Of Simulation Creation Toolkit Windows

In terms of pattern windows, one problem is the lack of consistency between the windows. For example, when selecting the agent, some pattern window buttons refer to the "Left Agent" and the "Right Agent", referencing the interacticon, and some actually have the terms for these agents i.e. "Agent To Count", for example, in the Data Pattern Window (see Appendix B.14). Furthermore, the Generate Pattern Window actually has a header at the top whereas the other windows do not (see Appendix B.13). This seems helpful in reminding the user of the pattern they are currently trying to implement (other windows have a small title in the menu bar). Ideally, these window traits would be made consistent over all the pattern windows; time constraints inhibited this from occurring, however, future iterations of the Simulation Creation Toolkit should make all the windows consistent.

In the Simulation Construction Kit Window (see Appendix B.1), the animation on the right hand side stops when the user clicks on a pattern specification. This is because the whole specification is no longer selected, but rather, just a part of the specification is selected (i.e. a checkbox the user just clicked for example). A few users were confused by this believing they had made a mistake. The fix for this problem is non-trivial; however, either a message should be given to the user alerting them to this fact, or the problem should be fixed in subsequent versions of the Simulation Creation Toolkit.

Originally, users also had the option of specifying a pattern in the pattern window itself. There are a few problems with this strategy. The first is that if a pattern has a huge number of specifications, like the Generate Pattern specifications for example, the size of the interacticon is compromised or the pattern window has to be made extremely large. The second problem with this strategy is that the pattern window becomes complicated. This

could make the selection of patterns a daunting task for students. Furthermore, the user may wonder why she/he has to specify the pattern twice leading to confusion. Thus, it was decided through meetings with my advisor, Dr. Alexander Repenning, that the user would only do the agent specification at the pattern window and the remainder of the specifications in the Simulation Construction Kit Window. Additionally, in order to alert the user to specify a given pattern they have just implemented, once a pattern is selected every open window in the Simulation Creation Toolkit closes except for the Simulation Construction Kit Window. At this point, the recently implemented pattern's specification appears selected in this window.

It should also be noted that after the user specifies a pattern, the user can minimize this specification box in the Simulation Construction Kit Window leaving only the name of the pattern and the agents involved; the user can also maximize this specifications box later if she/he needs to make changes to the pattern.

For the Pattern Picker Window, The Movement Picker Window, and the Collision Picker Window, it is not entirely clear that one must click on an animation to make a selection unless they read the text. This is in part because the animation gives the user no real clue that it is clickable. Furthermore, when a user does click on an animation, the next window pops up extremely quickly which could lead to confusion. The Simulation Creation Toolkit should somehow make it clearer that these animations are clickable, possibly by having the cursor change shape as it hoovers over them. Moreover, upon clicking, the animation should give the user some clue that it was selected; this could occur by changing the background color in the animation.

Finally, multiple windows in the Simulation Creation Toolkit have to be opened in order for a user to select a pattern. For example, to arrive at the Random Movement Pattern Window involves opening the Pattern Picker Window, the Movement Picker Window, and the Random Movement Pattern Window; thus, all three windows are open at the same time. These windows only close when the user actually adds the pattern (unless the user closes them manually). This could lead to a cluttered screen with many distracting interacticons. One solution to this problem would be to replace the current window with the next window corresponding to the users selection and adding a "back" button so the user can navigate back to the previous window.

For this study's purposes, the user interface proved adequate. However, in addition to the above changes, future research including many usability studies involving a variety of students could make the system more effective in the classroom environment.

3.8.2 Discussion Of Simulation Creation Toolkit Interacticons

Interacticons are animations that enact a given pattern. Like the Computational Thinking Patterns they represent, interacticons are a work in progress. For the most part, interacticons were found to be extremely useful in helping students identify a specific pattern. However, there are a few problems with how interacticons are presently implemented; these stem from the fact that interacticons are canned animations.

The first issue is that interacticons do not reflect a given users specification choices for a pattern. For example, if a user specifies that Agent A should generate Agent B to the left using the Generate Pattern specification, the interacticon for this specification continues to show Agent A generating Agent B to the right (see Appendix B.13). This can be confusing to

the user because the agents depicted in the interacticon are from their simulation, however, the interacticon does not reflect the current specification choice or the implemented AgentCubes code; thus, there is a disconnect between the interacticon shown and the simulation.

The second issue, related to the first, is that the agent behaviors in an interacticon do not reflect previously implemented patterns. For example, if the user implements Agent A generating Agent B, and the user previously implemented a Random Movement Pattern involving Agent B, the interacticon for the Generate Pattern should have Agent A generating a randomly moving Agent B (instead of Agent B moving directionally to the right). Again, this would better reflect the user's simulation.

Due to time constraints a solution for these two issues was not developed; however there are a few ways the Simulation Creation Toolkit could deal with these issues. The most challenging solution would involve making the interacticons reflect any given specification a user selects as well as any previous movement pattern the user may have implemented on a given agent. This could be accomplished by either making a different animation for a specification that would change the interacticon, or making the interacticon window itself a tiny AgentCubes world wherein the actual agent code is run but the only two agents in the world are the ones present in the pattern corresponding to the interacticon.

A much simpler (and possibly unsatisfying) solution might involve giving the user a message that indicated the interacticon might not be reflective of their previous pattern choices or current specification choices. The Simulation Creation Toolkit could also eliminate swapping the agents for generic disks; this might help the user better understand that the interacticon is merely a representative animation.

3.9 Looking Ahead

The thesis aims to investigate the use of Computational Thinking Patterns as a first step towards integration of simulation creation activities in the classroom. The Simulation Creation Toolkit seems like a promising avenue that could eventually accomplish this ideal. In a perfect world we would attempt to make a variety of students create as many different types of simulations as possible from the Guaranteed Viable Curriculum using the Simulation Creation Toolkit, and analyze the effectiveness of this tool in creating simulations as well as look more rigorously at how to improve the user-interface elements of the system. However, given time constraints, such studies must be included in further research. The subsequent chapters will instead explore and analyze one simulation building unit attempted with students: a Predator/Prey exercise that 6th and 7th grade students undertook using the Simulation Creation Toolkit. Furthermore, the subsequent chapters will also look at a small-scale study that explored how users with little guidance employed the system to create simulations by analogy based on high-level descriptions as to what the agents in the simulations should do.

CHAPTER 4

4. STUDY DESIGN

The Simulation Creation Toolkit was in part tested in the classroom environment at Centenary Middle School. However, because classroom units, for various reasons that will be discussed, necessitate a highly guided environment, an addendum study was performed. This study provided users (mostly college students) with three high level descriptions of programs and a brief introduction to the Simulation Creation Toolkit; these users then tried to create the simulation. This chapter will introduce both of these studies.

The Simulation Creation Toolkit was tested at Centenary Middle School with two teachers I had previously worked with while in the GK-12 program: Marco Cornacine and Marks Savs (see section 1.3, Marks Savs transferred from Neds High School to Centenary Middle School shortly after my GK-12 experience ended). Marco Cornichionne teaches 7th grade Life Science and Marks Savs teaches the 6th grade Exploratory Wheel Computer class. Marks Savs's class was included in the study to evaluate how students who had never used AgentSheets or AgentCubes before responded to using the Simulation Creation Toolkit.

Many factors have to be considered when integrating a study into a middle school classroom. Marco Cornichionne's syllabus is dictated by the Boulder Valley School District (B.V.S.D.) Guaranteed Viable Curriculum. The curriculum is written such that each topic is broken down into the weeks that it should be covered. Furthermore, students have to take a state mandated standardized test near the end of the school year to ensure that they are reaching certain standards. This restricts both the duration and the time of

year an unproven exploratory study, such as the Simulation Creation Toolkit study I proposed, can occur. Ideally, the teacher would want the study to happen after the state standardized test exam and with duration such that it would not infringe upon other topics that need to be covered. Additionally, for study effectiveness and to enable classroom integration, the study should attempt to teach subject matter that would be beneficial for students to garner a deeper understanding of previously covered topics.

In contrast to Marco Cornacine's Life Science class, Marks Savs's Computer Class has no such curriculum restrictions; the only big restriction in the Computer Class had to do with duration of the study. This is because the Computer Class is only 8 weeks long with different groups of 6th graders cycling through the class during the school year. Therefore, the study was tailored to meet the restriction of Marco's Life Science class.

To this end, I had many meetings with Marco Cornacine about the structure of the study and the types of topics that should be studied. Marco stated that the duration of the study could be 4 days in the month of April. He also outlined a range of topics that his students would be working through that quarter and were often misunderstood and/or hard to do hands on activities with. These included Predator/Prey, Photosynthesis, Global Warming, Evolution and Natural Selection. I picked the Predator/Prey simulation because I had previously done a Predator/Prey simulation in AgentSheets at Neds High School (see section 1.3). Marks Savs approved this unit too. However, given time constraints, Marks Savs could only do 3 days of study on the Simulation Creation Toolkit.

As mentioned above, an addendum study was also performed to understand to what effect users could use the Simulation Creation Toolkit to

program by analogy. This chapter will first describe the classroom study and then describe the analogical reasoning study in-depth.

4.1 Integrating Research Questions With In-Class Learning Targets

This study has two main goals. The first goal is to create the Simulation Creation Toolkit. The second goal is to answer the research questions laid out in the thesis. As stated above in section 1.6, these questions are as follows:

RQ1: Can students programming at the Computational Thinking Pattern level successfully create simulations of scientific phenomena they are presented within class?

RQ2: Does students programming science simulations using high level Computational Thinking Pattern lead to better student conceptualization and understanding of the material they are being taught?

Research Question 1 relates to whether students can successfully use the Simulation Creation Toolkit to create simulations. Generally, we are trying to answer if employing a high-level programming strategy at the Computational Thinking Pattern level is beneficial in facilitating student simulation creation. Research Question 2 looks at whether this strategy promotes students, through the act of creating and experimenting on the simulation, to better understand key concepts that should be conveyed through the study of this unit. As mentioned above, the Life Science Guaranteed Viable Curriculum at BVSD primarily dictates these concepts.

Research Question 1 can be measured by analyzing the correctness of the simulations students produce combined with evaluating answers to worksheet questions that target the ideas behind using this tool for simulation creation. Thus, the major instrument for Research Question 1 is how many patterns are implemented correctly in the simulations students create in addition to selected worksheet question answers. It should be reiterated that this does not technically mean that the simulation is a "more correct" representation than one that might have been wrongly implemented by a student—as stated in section 3.7, this thesis is concerned with a proof of concept indicating successful and easy student-creation of simulations in the classroom; though closely related, the ability to create realistic simulations and the degree of accuracy is something that should be covered in subsequent research. Therefore, student correctness is based on the ability to accurately implement patterns necessary to create a given simulation as interpreted by the instructor.

Research Question 2 delves into the usefulness of having students create in-class simulations at this high level. This question is more complicated to answer; however it is intimately related to the learning targets of a given unit. In terms of this study, we must not only try to gain insight into this research question, but also, see if we can use this idea of simulation creation to facilitate student understanding of often misunderstood concepts outlined in the Guaranteed Viable Curriculum. To put this another way, since this study takes place inside the Life Science classroom and uses a somewhat significant amount of time allotted for seventh grade students to study ecosystems and populations, necessitates the need for learning targets that not only help answer Research Question 2, but, as importantly, cover a significant amount of topics pertaining to this subject

matter as outlined in the curriculum. Furthermore, student creation and subsequent exploration of a simulation for this particular unit affords a unique opportunity to expose students to ideas in the curriculum that are otherwise hard for students to grasp using traditional classroom instruction.

To gain insight into Research Question 2, students completed worksheet questions as they created the Predator/Prey simulation. The relationship between Research Question 1, which aims to have students create a simulation using the Simulation Creation Toolkit, and Research Question 2, lies in how the completion of a given task using the Simulation Creation Toolkit helps students understand a given in-class concept. The ordering of worksheet questions relies heavily on the sequence these concepts are introduced in the simulation. Therefore, the curriculum ideas that students should be exposed to while doing this unit help shape both what the students do to complete the simulation and the concepts/worksheet questions they are exposed to as they progress through this simulation creation unit.

Also of note, as will be explained later in this chapter, students were given a tutorial to guide them through the simulation creation exercise. The worksheet questions happened to be integrated into this tutorial; therefore, for the remainder of this thesis the "tutorial" and the "worksheet" refer to the same document. The term "tutorial" will refer to the part of the document wherein instructions are given to students to complete the simulation. The term "worksheet" will refer to the questions posed to students throughout the document.

The following table summarizes how the Research Questions relate to the project methods and data collected.

Evaluation Objective	Outcome Measures	Data Sources	Method
RQ1	Pattern analysis of student-created Predator/Prey simulations (including specifications). Analysis of related worksheet questions.	6th grade computer class students (2 classes) and 7th grade Life Science students (4 classes).	Students create Predator/Prey simulation using tutorial and answering worksheet questions that gain insight into how well students understand the Simulation Creation Toolkit as they progress.
RQ2	Analysis of related worksheet questions.	Same as RQ1.	Students answer worksheet questions that target often-misunderstood concepts pertaining to the Predator/Prey unit in the Guaranteed Viable Curriculum. These questions are placed in the context of the simulation being created.

Table 6: How Research Questions Relate To Project Methods And Data

In developing a unit it helps to start with the specific learning targets that should be accomplished by students at unit completion. As stated in *Classroom Assessment For Student Learning* by Jan Chappius et. al.:

> "In standards-based schools what students are to learn drives all the planning, instruction and assessment. The curriculum documents are the roadmap we use and the assessment is the global positioning system that guides us to our destination[54]."

To better understand how the curriculum requirements helped shape this study, it helps to take a closer look at the 7th Grade Life Science Guaranteed Viable Curriculum as it pertains to this unit.

4.1.1 Related BVSD Guaranteed Viable Curriculum Requirements

As mentioned above, the BVSD Guaranteed Viable Curriculum outlines what students should learn over the course of the semester in a given class. There are two parts to the Guaranteed Viable Curriculum. The first part outlines the overarching questions that guide student study through each Life Science unit. The second part outlines the knowledge that students should gain through a particular unit. Both of these can be thought of as defining the "learning goals" for a given unit.

The organization of the Guaranteed Viable Curriculum learning goals can be separated into "overarching learning goals" and "supporting learning goals." According to Professor Erin Furtak, University of Colorado Boulder Department of Education, overarching learning goals can be thought of as follows:

> "An overarching goal…will orient you and your students toward what you ultimately want them to learn. . .what are often called "big idea" questions. These questions help to organize the information contained in an overarching learning goal into a question to drive student inquiry during your unit [55]"

There are multiple overarching learning goals in the Boulder Valley School District Life Science Guaranteed Viable Curriculum [56]. All of these overarching learning goals are labeled as five Science Standards in the Guaranteed Viable Curriculum; four of them relate to this particular unit.

Science Standard 1 requires students to "apply the process of scientific investigation and design, safely conduct, and communicate about and evaluate such investigations [56]." Science Standard 2 states that at the end of the class students should "know and understand common properties, forms, and changes in matter and energy [56]." In the Predator/Prey simulation this is in part manifested as "trophic levels" wherein energy exists

and is exchanged at each level of the food web (producers, consumers, decomposers etc.). Science Standard 3 states that students should "know and understand the characteristics and structure of living things, the processes of life, and how living things interact with each other and their environment [56]." In terms of the Predator/Prey simulation, students can have the opportunity to explore how prey populations affect predator populations and vice versa. Furthermore, students can gain insight into how external factors, like poaching, affect this relationship. Finally, Science Standard 5 states that students should "understand that the nature of science involves a particular way of building knowledge and making meaning of the natural world [56]." As stated in Chapter 1, one way students can build knowledge is to create and experiment with simulations (see section 1.3). In essence, this project is an investigation into the possibility of using the Simulation Creation Toolkit as a way to facilitate this knowledge building by enabling students to create simulations in the classroom.

These Science Standards are extremely general; the Guaranteed Viable Curriculum also includes an overview as to the topics and concepts that the Life Science class should cover as well as specific knowledge students should gain in each unit of study. The following picture shows the content areas that the Life Science class should cover over the course of the school year; the areas that this four day unit has the potential to touch upon are colored in blue.

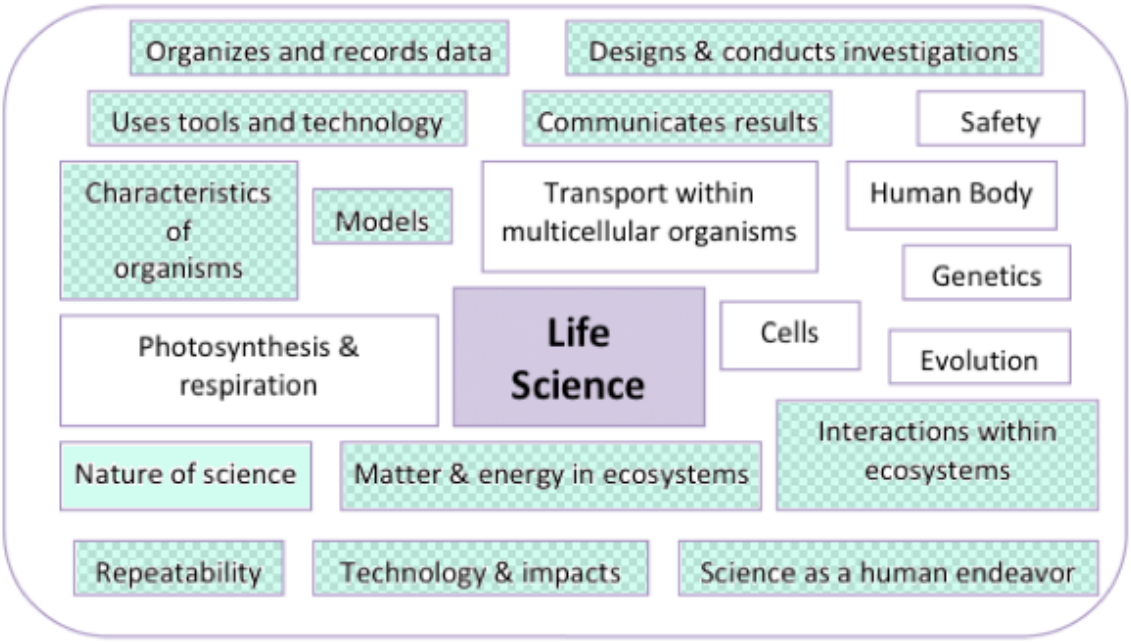


Figure 80: BVSD Life Science GVC Topic Outline With Relevant Topics Colored In Blue [56]

As shown in Figure 80, enabling the use of simulation creation in the classroom has the ability to touch on many of the Life Science curriculum topics in a relatively short amount of time. However, the study design must be created such that these topics are addressed. The challenge of this study, therefore, is to try to cover as many of these topics as possible in the short amount of time given while trying, at the same time, to answer Research Question 1, which necessitates that students are able to create the given Predator/Prey simulation. This will be discussed further later in this section.

As mentioned above, in addition to overarching learning goals, Furtak discusses supporting learning goals. Specifically Furtak describes supporting learning goals as follows.

> "Supporting goals build on each other toward the overarching goal and big idea questions for your unit, forming the steps that will help your students climb as they come to know the science content you want them to learn [55]."

In the context of how this study fits into the Guaranteed Viable Curriculum there are also many possible supporting learning goals that relate to this unit. These are referred to as "Essential Learnings" in the Guaranteed Viable Curriculum. We will now briefly outline the Essential Learnings and relate these to the high level Science Standards they fall under.

As mentioned above, Science Standard 1 in part states that "Students apply the processes of scientific investigation and design…(and) communicate about and evaluate such investigations." In the context of this unit, the Essential Learning that Marco Cornacine and I decided we would try to convey were LS-2 and LS-3 (the Essential Learnings are denoted with LS and a number. A letter is used after the number to denote further subcategories of these essential learnings. For example, LS3-A would be the "A" subcategory of the LS3 essential learning). LS2 states that a student "Accurately uses appropriate tools and technology and metric measurement units to gather, organize, and analyze data and report results [56]." The subcategories of this Essential Learning we decided to concentrate on were LS2-A which requires that the student "Records reports and analyzes data in a variety of forms…", and LS2-C which in part states that a student "Collects organizes and interprets data…[56]." LS3 states that a student "Interprets, analyzes and evaluates data and recognizes bias in order to formulate logical conclusions [56]." Given that students are creating a simulation to experiment on it is not surprising that this Learning Standard would be targeted. The subcategories of LS3 that we decided to focus on were LS3-A, LS3-B, LS3-C, and LS3-D. LS3-A states that a student "Interprets analyzes, and evaluates data/observations…to formulate logical conclusions [56]." LS3-B requires that a student "Uses evidence to state if a hypothesis is supported or not supported [56]." LS3-C states that a student "Makes predictions based

on experimental data [56]." Finally LS3-D states that students should "State explanations that link claims and evidence [56]." Achieving any of these subcategories will reinforce the Essential Learning helping students accomplish Science Standard 1.

Science Standard 2 in part requires that students learn the "common properties and forms and changes in matter and energy [56]." For this unit Marco Cornacine and I came to the conclusion that the Essential Learning we would try to convey is LS9 which states in part that a student "Explains how matter cycles and energy flows through ecosystems…[56]" Specifically, we decided to focus on LS9-G: "(students) infer the number of organisms or amount of energy available at each level of the energy pyramid [56]." This Predator/Prey simulation unit provides an opportunity to discuss trophic levels in an ecosystem and make the idea of energy and matter transferring between trophic levels more concrete. Introducing this idea enables students to take a step towards achieving Science Standard 2.

Science Standard 3, in part, requires students know "how living things interact with themselves and the environment [56]." For this unit Marco and I decided to concentrate on LS12 which states the student "Analyzes implications of interactions among organisms, populations, and their environment [56]." Specifically, we decided to focus on LS12-A and LS12-B. To accomplish LS12-A students must be able to "describe several factors that could limit the size of a population [56]." In the context of the predator prey simulation, students can analyze changes in breeding, limiting the initial food supply of the Predator/Prey, and/or limit parameters like the speed at which the Predator/Prey move among many other things. LS12-B states that students must be able to "describe the impact of humans on the environment and how that affects survival of populations and entire species

[56]." Upon deciding that we would implement a Predator/Prey simulation unit in the class, Marco asked that a Poacher Agent be included so the students could add an example of external human factors having an effect on their populations.

Science Standard 5 requires students to "understand that the nature of science involves a particular way of building knowledge and making meaning of the natural world [56]." Marco and I decided that this unit would focus on LS15 and LS15-A. LS15 states that students should "create and use physical and conceptual models for explanation and prediction [56]." LS15-A states that students "should recognize that models can be used to obtain information about processes that would be otherwise hard to study [56]." Furthermore, we decided that we should also focus on critical thinking as it pertains to models—i.e. where a particular representational system might break down or be unrealistic. Though not an explicit learning goal, the Guaranteed Viable Curriculum does cover this idea in its "Enduring Understanding" section in Science Standard 5 wherein it states "(students should learn) a model is something similar to but not exactly like what is being modeled. Some models are physically similar to what they are representing while others are not [56]." Given that this unit is a modeling unit, Science Standard 5 is closely related to this study.

4.2 Arriving At A Unit That Integrates RQ1 and RQ2 With GVC Learning Targets

As mentioned above, RQ1 deals with the ability of students to create a given simulation. For the purposes of this unit, students created a Predator/Prey simulation. RQ2 deals with whether the specific strategy for simulation creation, outlined thus far in this thesis, actually helps student

understanding of key concepts that should be learned through studying this unit. The general strategy for unit creation consisted of first outlining the steps that students must accomplish to successfully create this simulation and then placing appropriate worksheet questions at various steps of this simulation creation that not only help to assess the aforementioned learning targets, but also, is contextualized based on what the students have accomplished to that point in the unit creation activity. This section will review the steps students had to take to correctly create the simulation (related to RQ1), the general learning goals that Marco and I determined could be targeted at each step (related to RQ2 and the GVC), and finally will present the worksheet that was used to both help students create the simulation and gain insight into the concepts students did and did not understand.

4.2.1 The Simulation Set-Up

The Predator/Prey simulation is generally outlined in 3.7.1. The Predator/Prey simulation created in this unit employs 16 patterns with multiple specifications (see section 3.7.1). To the degree students successfully create these 16 patterns gives insight into Research Question 1. The pattern descriptions and specification choices often have to refer to agents and depictions in the simulation, so before delving into the patterns used to create the simulation, the general simulation setup will be described.

Students were given an AgentCubes program with all the pre-created agents necessary for the Predator/Prey simulation. Given the time constraints, it was decided that having students interact with the Simulation Creation Toolkit as quickly as possible would be better than having them take time creating agents (creating agents can be fun for students as they are

able to create the depictions for these agents, but also, is notoriously time consuming, and is an AgentCubes task more than a Simulation Creation Toolkit task). Furthermore, three Data Patterns that counted the number of Fox Agents, Rabbit Agents, and Grass Agents in the simulation were created beforehand in the simulation given to students (see Data Pattern in section 3.6.10). These patterns are not included in the 16 patterns students had to create.

What follows is a brief description of the agents provided to the students to complete their simulation. The following figure shows the six agents used in this Predator/Prey simulation.

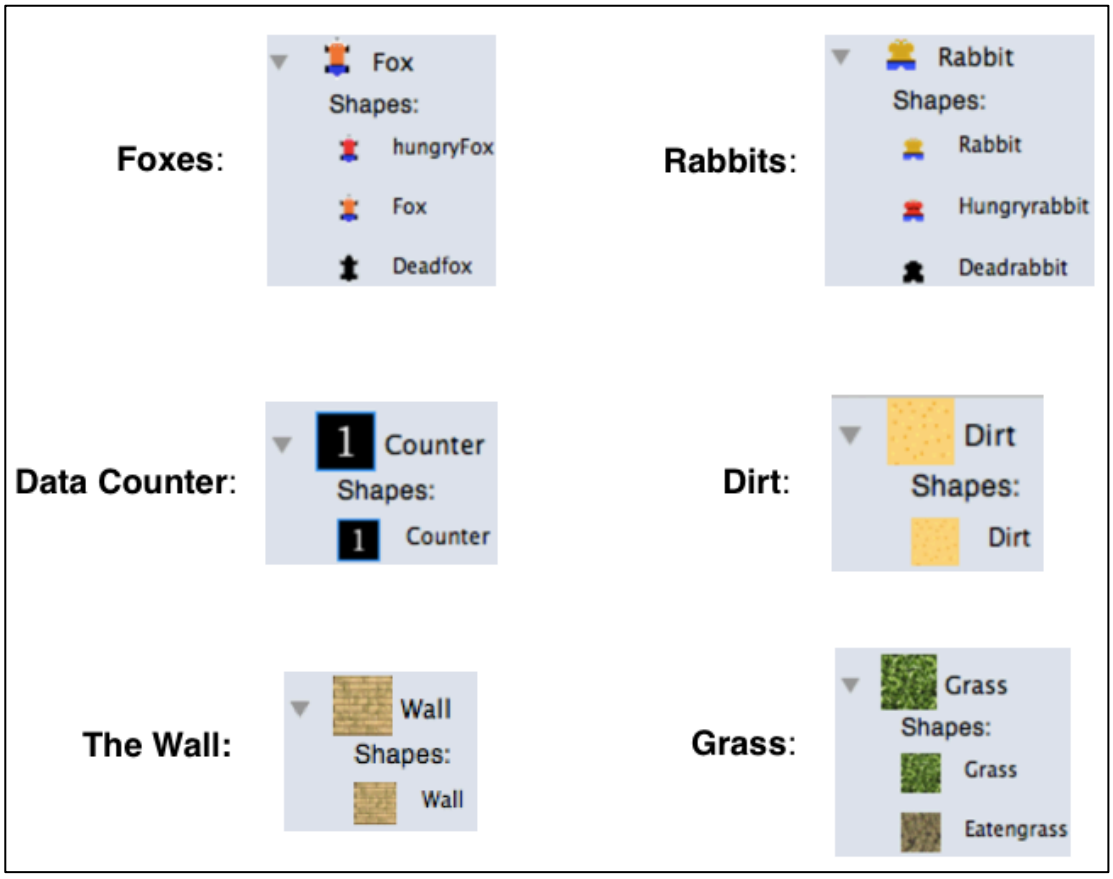


Figure 81: The Agents Used In The Predator/Prey Simulation

The top left of Figure 81 shows the Fox Agent with its three shapes (in order from top to bottom): the Hungryfox (red), the normal Fox (tan), and Deadfox (black). To the right of the Fox Agent in Figure 81 is the Rabbit Agent. Similar to the Fox Agent, the Rabbit Agent has three shapes (in order from top to bottom): the normal Rabbit (off white), Hungryrabbit (red), and Deadrabbit (black).[7] Directly under the Fox Agent is the Counter Agent which keeps track of the number of each agents in a world (see section 3.6.1); it has only one depiction. To the right of that agent is the Dirt Agent. This is the background agent for this simulation. Below the Counter Agent is the Wall Agent. The Wall Agent encloses the world and enables the Dirt Agent to always be to the left of any Rabbit or Fox agent. The reason why this is important will become clearer later in this section. Finally, in the bottom right of Figure 81 is the Grass Agent. This agent has two depictions (from top to bottom): regular Grass and Eatengrass.

Finally, two worlds were provided for the students. One world, which was populated with the minimal amount of agents, was provided to students so they could see if an implemented pattern happened to work. This world looks as follows and will be referred to as the "Test World" for the remainder of this thesis.

[7]The Fox and Rabbit Agents were graciously created by Hunter Stevens, undergraduate student at the University Of Colorado Boulder.

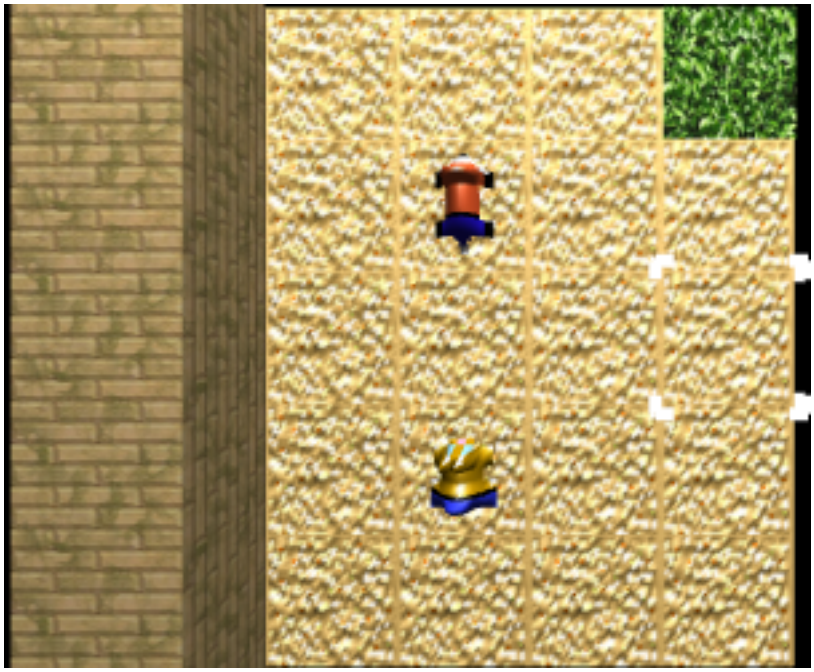


Figure 82: The Test World Provided To Students

The Test World depicted in Figure 82 is a 5 by 5 level covered with a layer of Dirt Agents along its whole surface. Wall Agents are stacked upon these Dirt Agents on the left perimeter of the world. In the top right of the world is a normal Grass Agent. Finally a Rabbit Agent and A Fox Agent appear in the middle of the level. This simple level enabled students to see if a specific pattern worked without having to decipher a huge number of agents at the same time. For example, the student might want to test if the Fox Agent changes the regular Rabbit Agent into a Deadrabbit Agent, but might not be able to detect when or if this occurs in a highly populated AgentCubes world.

The other world provided to students will be called the "Simulation World" for the remainder of the thesis. This world looks as follows.

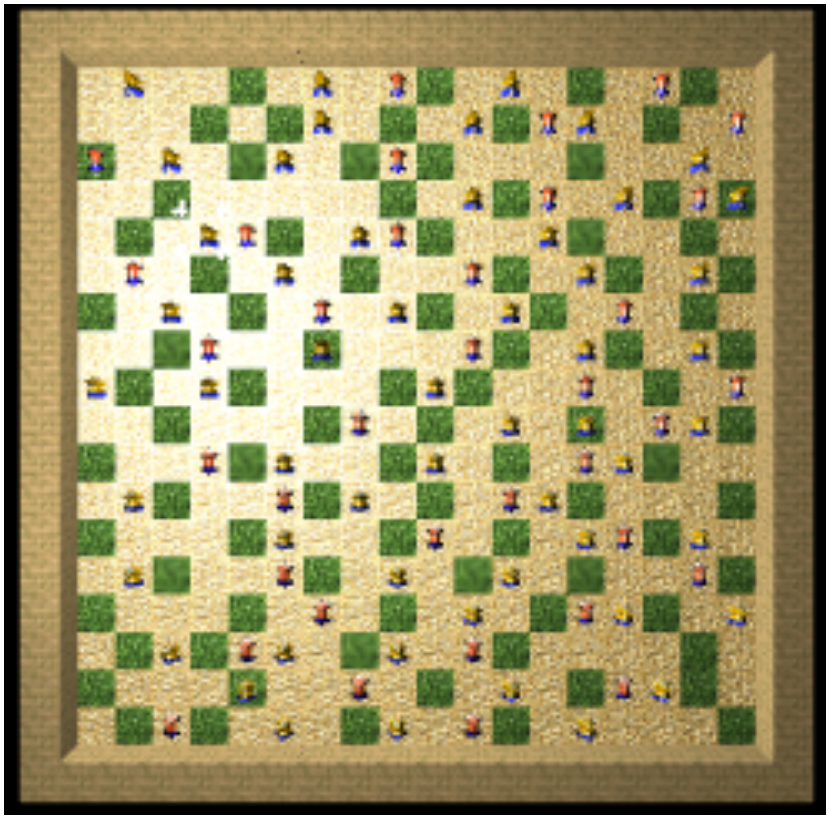


Figure 83: Simulation World Provided To Students

The Simulation World is a 20 by 20 level covered with a layer of Dirt Agents and encircled by Wall Agents. It initially contains 36 Fox Agents, 54 Rabbit Agents, and 89 Grass Agents, all with their normal shapes.

The 16 patterns and specifications for each pattern, with an explanation of what they accomplish in this simulation are as follows. These patterns are presented in order of how the students were instructed to implement them though pattern implementation order is theoretically not important in the Simulation Creation Toolkit (see section 3.3.3). The general pattern descriptions and specifications for each of these pattern implementations can be found in section 3.5 and 3.6. Finally, throughout this section, if, for example, we must refer to the "Fox Agent with the Hungry Fox shape", we will instead say the "Hungry Fox Agent." This is done for simplicity and does not mean that the Hungry Fox Agent is a

different agent than then the regular Fox Agent; refer to Figure 81 if confusion arises.

1) **Pattern:** Fox Agent randomly moves at a certain speed. This uses the Random Movement Pattern.

   Description: Enables the Fox Agent to move randomly around the world.

   Specifications: The speed of the Fox Agent defaults to 0.5 times per second and students were not instructed to modify this specification. The Fox Agent is blocked by 7 Agents: The Wall Agent, all three Fox Shapes and all three Rabbit Shapes; it is up to the students to select these agents correctly. Note that we do not specify "for all agents"—the Hungry Fox Agent will track the Rabbit (and the Dead Fox Agent should not move).

2) **Pattern:** Rabbit randomly moves at a certain speed. This uses the Random Movement Pattern.

   Description: Enables the Rabbit Agent to move randomly around the world.

   Specifications: Like the Fox Agent, the speed of the Rabbit Agent defaults to 0.5 and is not modified by students. The Rabbit Agent is also blocked by 7 Agents: The Wall Agent, all three Fox Shapes and all three Rabbit Shapes; it is up to the students to select these agents correctly.

3) **Pattern:** Dirt Agent changes regular Fox Agent into a Hungry Fox Agent to the right with a given percent chance. This uses the Change Pattern.

   Description: The Fox Agent should get hungry over time. In past Predator/Prey simulations we did this by counting the number of steps

the Fox Agent took, and if it reached a certain threshold, the Fox Agent would get hungry. In this simulation, it was thought that having the Dirt Agent, which exists over the whole level, make the Fox Agent change into a Hungry Fox Agent might be easier to implement for students and would have the same general effect. Furthermore, this is why the level is surrounded by Wall Agents; namely the Fox Agent cannot move all the way to the left of the level meaning that the Dirt always has an opportunity to change the Fox Agent into a Hungry Fox Agent.

Specifications: Once every 2 seconds there is a 40% chance that a Dirt Agent will change a regular Fox Agent into a Hungry Fox Agent to the right.

4) **Pattern:** Dirt Agent changes regular Rabbit Agent into Hungry Rabbit Agent to the right with a given percent chance. This uses the Change Pattern. The description and specification is identical to pattern 3 except the Dirt Agent changes the Rabbit Agent into a Hungry Rabbit Agent instead of a Fox Agent into a Hungry Fox Agent.
5) **Pattern:** Dirt Agent changes Hungry Fox Agent into Dead Fox Agent to the right with a given percent chance. This uses the Change Pattern.

   Description: If the Hungry Fox Agent goes enough time without food, the Hungry Fox Agent should die. Similar to pattern 3, in the past this would occur with a certain number of steps without encountering food (i.e. a prey agent); however, given that the Simulation Creation Toolkit does not have this functionality, accomplishing this by using the Change Pattern with the Dirt Agent works as well.

Specifications: Once Every 2 seconds there is a 40% chance that a Dirt Agent will change a Hungry Fox Agent into a Dead Fox Agent to the right.

6) **Pattern:** Dirt Agent changes Hungry Rabbit Agent into Dead Rabbit Agent to the right with a given percent chance. This uses the Change Pattern. The description and specification is almost identical to pattern 5 except with the Hungry Rabbit and Dead Rabbit Agents taking the place of the Hungry and Dead Fox Agents.
7) **Pattern:** Dirt Agent changes Dead Fox Agent into Grass Agent to the right with a percent chance once every so often. This uses the Change Pattern.

   Description: To simulate decomposition it was decided to have the Dead Fox Agents stay on the screen for a little bit of time and then to be replaced by a Grass Agent to represent the plant growth post decomposition.

   Specifications: Dirt Agent changes the Dead Fox Agent into a Grass Agent to the right with a 100% chance once every 4 seconds.
8) **Pattern:** Dirt Agent changes Dead Rabbit Agent into Grass Agent to the right with a percent chance once every so often. This uses the Change Pattern. The description and specification is almost identical to pattern 7 except with Hungry Rabbit and Dead Rabbit Agents instead of the Hungry and Dead Fox Agents.
9) **Pattern:** Hungry Fox Agent tracks the Rabbit Agent. This uses the Tracking Pattern.

   Description: The Predator/Prey model students typically program in prior AgentSheets units involved hungry predator agents chasing prey agents to eat them. Similarly, for this unit students implement the

Tracking Pattern when the Fox Agent chases the Rabbit Agent. The Fox Agent is allowed to move on both the Grass Agent and the Dirt Agent to track the Rabbit Agent (i.e. it does not walk over other Fox Agents to get to the Rabbit).

Specification: The Hungry Fox Agent tracks the Rabbit Agent once every .5 seconds, for all depictions of the Rabbit Agent, and is allowed to move on Dirt Agents and Grass Agents. It should be noted that it is possible that a Dead Rabbit Agent will be tracked; this is probably unrealistic but works for our purposes.

10) **Pattern:** Hungry Rabbit Agent tracks regular Grass Agent. This uses the Tracking Pattern. The description and specification is almost similar to pattern 9 except that the Hungry Rabbit agent does not track all Grass Agents (i.e. the Rabbit Agent does not track the Eaten Grass Agent) and only moves on the Dirt Agent (if the Hungry Rabbit Agent encounters grass, it will no longer be hungry).

11) **Pattern:** Hungry Fox changes the Rabbit Agent into a Dead Rabbit Agent. This uses the Change Pattern.

Description: This pattern represents the Hungry Fox Agent eating the Rabbit Agent. Note that this works for both the regular Rabbit Agent and Hungry Rabbit Agent (it also works for the Dead Rabbit Agent but the user will not actually see a change in this case).

Specification: In any direction, for all depictions of the Rabbit Agent, with a 100% chance once every 0.0 seconds, the Hungry Fox Agent changes a Rabbit Agent of any shape into a Dead Rabbit Agent.

12) **Pattern:** Hungry Rabbit Agent changes the Grass Agent into the Eaten Grass Agent. The description and specification for this pattern is similar to Pattern 11 with the Hungry Rabbit Agent taking the place

of the Hungry Fox Agent and the Grass Agent taking the place of the Rabbit Agent.

13) **Pattern:** Dead Rabbit Agent changes the Hungry Fox Agent into regular Fox Agent. This uses the Change Pattern.

    Description: The Hungry Fox Agent changes the Rabbit Agent into a Dead Rabbit Agent to represent the Hungry Fox eating the Rabbit Agent. At this point, the Hungry Fox Agent should change back into a regular Fox Agent; the Dead Rabbit Agent accomplishes this change. This change happens in any direction, which has a few implications. The first is that any Hungry Fox within 1 space of the Dead Rabbit in any direction will be changed back into a Regular Fox Agent. The second is that, given that this is the case, the Hungry Fox Agent that actually changes the Rabbit Agent into a Dead Rabbit Agent (by eating it) is not the only Fox Agent nourished by that Rabbit Agent. It is debatable how realistic or unrealistic this is, but it should be noted nevertheless.

    Specification: The Dead Rabbit Agent changes the Hungry Fox Agent back into a regular Fox Agent once every 0.0 seconds with a 100% chance in any direction.

14) **Pattern:** The Eaten Grass changes the Hungry Rabbit Agent into a normal Rabbit Agent. This uses the Change Pattern. The description and specification is the same as pattern 12 with the exception that the Hungry Rabbit Agent takes the place of the Hungry Fox Agent and the Grass Agent takes the place of the Dead Rabbit Agent.

15) **Pattern:** If a normal Fox Agent sees another normal Fox Agent in a certain direction, a new Fox Agent is generated in another direction. This uses the Generate Pattern.

Description: This pattern simulates mating. Having the Fox Agent see another Fox Agent in any direction would require tweaking of the mating rates to stop rapid overpopulation so just one direction is used. In this simulation, the Fox Agent can create another Fox Agent on top of a Fox Agent that already exists; the pattern as of yet does not allow the Fox Agent to see something in two directions in order to generate (see section 3.6.9). Hungry Fox Agents cannot mate; how realistic this may be is debatable. Furthermore, it was decided to not have two different sexes of Fox Agents as it would have increased the number of Fox shapes from 3 to 5 making the simulation more complicated to complete. However, not including male and female shapes has often been used in prior AgentSheets Predator/Prey modeling exercises.

Specification: Once every 1.5 seconds with a 50% chance, if a normal Fox Agent sees another normal Fox Agent to the right, it creates a new Fox Agent in the downward direction.

16) **Pattern:** If a normal Rabbit Agent sees another normal Rabbit Agent in a certain direction, a new Rabbit Agent is spawned in another direction. This uses the Generate Pattern. The description and specification for this pattern are similar to pattern 15 with the exception that the Rabbit Agent mates with a 75% chance.

The above patterns are what students had to complete to get the full simulation correct. The degree to which students were able to implement these 16 patterns over the allotted time helped answer Research Question 1. In addition to these patterns, the tutorial/worksheet included an experiments section. In this section students created a Poacher Agent and a Bullet Agent and had the Poacher Agent generate Bullets using the Generate Pattern, the

Bullet Agent move using the Directional Movement Pattern, and finally, had the Bullet Agent absorb Fox Agents using the Absorb Pattern (or change the Foxes into Dead Foxes using the Change Pattern).

Given that the students implemented these patterns, let us now look at the worksheet given to students to better understand what assessment questions were asked, and where in this simulation they were posed.

4.2.2 Introduction To Predator/Prey Unit Worksheet Questions

This section and the next will describe the worksheet students used as they created the Predator/Prey simulation outlined in section 4.1.1. As mentioned above, the worksheet was written to get some insight into the Research Questions and address content areas required by the Guaranteed Viable Curriculum (see section 4.1). It should be noted that a few of the questions are meant to gain insight into Research Question 1 too; namely questions that get at whether students understand the patterns and system parameters being implemented. Furthermore, given the aim of Research Question 1, to see if students could actually complete the simulation, the worksheet questions had to be limited by the total amount of time for the unit.

It was decided that the worksheet questions would be integrated into the tutorial for the simulation handed out to students. The questions themselves, therefore, had to be quick to answer, and yet, had to relate to Guaranteed Viable Curriculum concepts Marco and I thought might be helpful to cover. It should be noted that the tutorial takes students step by step through the simulation creation process; the merits of such a strategy will be discussed in section 4.2.4. The full tutorial can be found in Appendix C; all the questions will be discussed in section 4.2.3.

The tutorial is split up into five days. However, these were just guidelines and most students worked ahead of the schedule. At the end of day 4, the students would have completed the 16 patterns outlined in section 4.2.1. Day 5 focused on experimenting on the simulation students created by changing parameters (i.e. pattern specifications) and adding a Poacher Agent. Day 5 is more open-ended with less step-by-step instruction of how to accomplish certain patterns using the Simulation Creation Toolkit. This was done intentionally to see how far students could get with little or no scaffolding.

Feedback that helped shape the tutorial was provided by Jeff Hoel, Ph.D. Department of Computer Science University of Colorado Boulder and Professor David Webb's class EDUC 5706 Assessment in Mathematics and Science, Spring 2012. David Webb is a Professor in the Department of Education at the University of Colorado Boulder.

Research Question 2 is challenging to answer for multiple reasons. First, to make a claim that the act of simulation creation actually helped student understanding pertaining to a particular topic implies that students did not understand a concept, programmed the simulation, and then, after the programming exercise, better understood the concept. A better way to measure this question would have been to give students a pre-test, see what questions or concepts were readily misunderstood, and then retest students post simulation creation. Time constraints caused by both system creation, Internal Review Board scheduling, and duration of the study in the classroom itself inhibited this strategy.

Given that the above described pre/post test strategy is not employable, another strategy exists to indicate if users may have a deeper understanding based on the task they completed using the Simulation

Creation Toolkit. This strategy has previously been employed by Professor Clayton Lewis, Department of Computer Science at The University Of Colorado Boulder [57]. The strategy employs elements of the Cognitive Walkthrough and involves looking for questions that highlight "Signature Phenomena." Signature Phenomena in this context is focused on whether a student is better equipped to answer a particular question as a direct result of the simulation steps they just accomplished. Specifically, this analysis involves analyzing each question, before looking at how students themselves answered the questions, and appraising each question according to two "phases". The two phases, contextualized for this study, are as follows.

Phase 1: What answers would indicate understanding?

Phase 2: What can one say about the process of simulation creation helping to answer the question.

Employing these phases to analyze each question can provide insight into the task of creating the simulation and the ability of the students to answer the questions posed to them..

We will now review all the relevant worksheet questions. For the purposes of analyzing each worksheet question, we will first present the question and discuss how the question relates to this project's Research Questions as well as the Guaranteed Viable Curriculum, and then, use the two above phases in order to better understand the relationship between the question and simulation creation. Finally we will end with a general discussion of these questions and their significance in answering the Research Questions.

4.2.3 An Analysis Of Predator/Prey Unit Worksheet Questions

Appendix C contains all the tutorial/worksheet materials. As mentioned in Appendix C, the tutorial and worksheet were combined and with worksheet questions placed at appropriate points. For information on the Guaranteed Viable Curriculum Science standards refer to section 4.1.1. Phase 1 and Phase 2 are described in section 4.2.1; Phase 1 will also serve to outline the possible "correct" answers for these questions meaning answers that indicate understanding. A rubric for these questions will be presented with the results in Chapter 5.

Question 1,2.

Description: In Question 1, students are asked to run the simulation, open simulation properties, and record the initial amounts of Grass Agents, Rabbit Agents, and Fox Agents. In Question 2, students open the Simulation Creation Toolkit and see that three Data Patterns have already been implemented keeping track of the agent populations in simulation properties. The aim of this question is to help students realize immediately that these patterns they are implementing add code to the simulation.

Relationship To Research Questions: This question relates tangentially to Research Question 1 as it gives us insight as to how quickly students are able to grasp the ideas behind the Simulation Creation Toolkit. As will be discussed later in the chapter, at this point of the exercise students have been given a 15-20 minute introduction on patterns and how they relate to AgentCubes. If students are able to pick up the idea that the populations of agents being counted and these three already implemented patterns are connected, it bodes well for students being able to use the Simulation

Creation Toolkit with minimal instruction; however, this may be a completely unrealistic expectation.

Relationship To GVC: This question acts as an introduction to Science Standard 5 and specifically LS15 because it sets the foundation for the initial agent populations in the simulation enabling students a basis of comparison once the actual simulation creation and subsequent simulation runs begin. Furthermore, it relates to Science Standard 1 and LS2-A wherein students record data.

Phase 1: The correct answer is 36 Fox Agents, 54 Rabbit Agents, and 89 Grass Agents.
Phase 2: This is not applicable as students have not begun the creating the simulation at this point.

Question 3.

Description: After implementing the Random Movement Pattern for the Fox and Rabbit Agent, students are asked why this movement might be unrealistic.

Relationship To Research Questions: This question has tangential links to Research Question 1 in that students can understand the differences between their simulation and reality. Given that students realize they are not creating an exact replica of a given phenomena, but rather, a representational system aimed at increasing understanding or solving a given problem, may help them better create simulations using the Simulation Creation Toolkit. This understanding is a first step towards abstracting out the interactions of a

given phenomena and mapping them into the context of the Simulation Creation Toolkit. Furthermore, this question marginally relates to Research Question 2 in that students must understand that animals probably do not move randomly around when they are not hungry.

Relationship To GVC: This relates to Science Standard 5 and thinking critically about where the representational system of the simulation breaks down. This also relates to Science Standard 1, specifically LS3-A wherein students interpret and evaluate based on observation.

Phase 1: The answer that would indicate understanding would be something akin to Rabbits and Foxes do not move randomly around all day.

Phase 2: The process of simulation creation, thus far, does not help at arriving at the above Phase 1 answer. Students have implemented two patterns, but the fact that their agents are now moving randomly does not in an of itself give students any inclination as to whether this movement is realistic or not. Rather, students have to retrieve previous knowledge of animal behavior, in addition to this simulation, to determine that this movement is unrealistic.

Question 4.

Description: Question 4 tells the students that we will now implement the Dirt Agent changing the regular Fox Agent and the regular Rabbit Agent into the Hungry Fox Agent and the Hungry Rabbit Agent respectively. It then asks them what pattern they would use to implement this.

Relationship To Research Questions: This question heavily relates to Research Question 1 in that if students, in general, are able to figure out the pattern to use to create the phenomena described, then the Simulation Creation Toolkit might be an effective strategy for integrating simulations into the classroom.

Relationship To GVC: At a high level one can think of being able to determine the pattern necessary to use as an exercise in abstraction that enables students to create representational systems. In this sense, it is partly related to Science Standard 5.

Phase 1: The correct answer to this question would be the Change Pattern, or at least, a description of the Change Pattern.

Phase 2: The process of simulation creation helps answer this question but in a very specific sense. This question is about which pattern to use to implement a certain phenomena using the pattern construct introduced by the Simulation Creation Toolkit. The correct answer could mean that students are correctly abstracting out the given interaction and finding ways to implement this interaction using the Simulation Creation Toolkit. However, it does not give any insight as to whether students have garnered an understanding of Predator/Prey interactions as a whole.

Question 5.

Description: Question 5 is posed after the Dirt Agent changes the Rabbit and Fox Agents into Hungry Rabbit and Fox Agents, and asks what is being used up as Foxes and Rabbits move; this question gets at the high level idea of

energy usage. A better way to phrase this would have been what is being expended that allows these animals to move. However, in the context of the Life Science class, this question makes sense as the idea of energy being expended had previously been covered in-depth.

Relationship To Research Questions: This has a slight relationship to Research Question 2 as programming the simulation and actually seeing the Fox and Rabbit Agents move could make the idea of energy consumption more concrete. However, this might be a stretch as it is much more likely students recalled this idea from past lectures.

Relationship To GVC: This question relates to Science Standard 2 and specifically LS 9-G wherein students begin to see the energy consumption of organisms.

Phase 1: A correct answer to this question would refer to energy being consumed as the animals move around. This answer requires an abstraction from the simulation to the real world wherein animals would actually expend energy and thus, requires students understand the agents as being representations of real-world animals.

Phase 2: Simulation Creation probably does not help in answering this question. Students will either know or reason that energy is expended as animals move or they will not; actually making the Rabbit and Fox Agents move in this simulation does not necessarily help a student understand this concept thought it might make this concept more concrete.

Question 6.

Description: Question 6 asks whether the regular Fox and Rabbit Agent are more or less likely to get hungry as the simulation runs, and further asks if this is realistic. This is another energy related question but taken a step further—if an animal needs energy it might want to seek out food.

Relationship To Research Questions: This question relates both to RQ 1 and RQ 2. If the student has understood what they have created, or created the simulation correctly and correctly interpreted it while test running the program, they can get the correct answer to this question. This relates to Research Question 1 because it involves the student correctly understanding the simulation they have created. This question is also a basic example as to how simulation creation can help facilitate understanding; namely, the student could create the simulation, play around with it, and see that as time passes and the agents move, the agents are more likely to get hungry. This relates to Research Question 2, as in this case, students would use the simulation to gain a deeper understanding of the material.

Relationship To GVC: Like the previous question, this question also relates to Science Standard 2 and LS9-G as we are looking at energy consumption in organisms. It also relates to Science Standard 5 and LS15 as students are analyzing the model and possibly gaining a better understanding of how hunger in this model is represented.

Phase 1: A correct answer to this would mention that as the simulation goes on, the Fox and Rabbit Agents are more likely to get hungry. A correct answer to follow up question would state that this is, in fact, realistic

behavior as the longer an organism goes without nourishment the more likely it is that they will need nourishment.

Phase 2: There are two approaches a student could have to answering the first part of this question. One is that the student a priory understood the link between energy consumption, movement and food, and thus, answers the question from memory. The other possibility is that the student runs the simulation, notices the trend of Fox and Rabbit Agents eventually getting hungry as they move around, and thus, figures out that these agents are in fact more likely to get hungry as the simulation continues on. This second possibility is enhanced significantly by the correct programming of the simulation up to this point and would be an indication that simulation programming may enhance understanding of certain concepts. However, given a correct answer, it is impossible to know which method (or combination thereof) the student used to get to that conclusion.

The follow up question is completely based on students already knowing that animals get hungry as energy is expended, and therefore, programming the simulation does not necessarily help answer this question, but rather, reinforces what they may have already learned in class.

Question 7.

Description: Question 7 asks that instead of stop moving what should the Hungry Fox and Hungry Rabbit do. This question is placed in between where the students implement the Fox and Rabbit Agents becoming hungry but before they implement the Tracking Pattern for both of these hungry agents.

Relationship To Research Questions: This question is meant to motivate students to think about what still needs to be completed in our simulation. This questions could have marginal links to Research Question 1 if the students use the term "tracking" or describes the Tracking Pattern based on the fact that they heard the pattern name in the class introduction. If students are able to recall the pattern names or describe the patterns and apply them to behaviors they want the agents to enact, then the student may be more likely to make the link between that pattern and other scientific phenomena that uses the same pattern. It also has marginal links to Research Question 2 if the student understands from programming the simulation up to this point that the Hungry Fox Agent and Hungry Rabbit Agent not moving at all seems unrealistic especially given the energy expenditure idea presented in the previous questions. In that case the student might use the context of the simulation plus their previous knowledge to come to the correct conclusion that at this point, for example, the Fox Agent should seek out the Rabbit Agent.

Relationship To GVC: Like the previous question, this question also relates to Science Standard 2 and LS9-G as we are looking at energy consumption in organisms but now are integrating the knowledge we have of energy consumption and hunger to guide us into the next phase of simulation creation. Therefore it also relates to Science Standard 5 wherein we take phenomena from science and contextualize it in the domain of the model we are creating. Finally, this question also relates to Science Standard 1 and LS3-A and LS3-C because students are interpreting what they are observing from the simulation, realizing wherein the simulation might be unrealistic,

and then making a prediction as to what should be implemented in the future based on these observances.

Phase 1: The correct answer would be that the Hungry Fox Agent should chase the Rabbit Agent, and the Rabbit Agent should track the Grass Agent. Other acceptable answers would be obtain food etc.

Phase 2: Programming the simulation up to this point does not necessarily help you answer this question per se. It does help motivate the question in that, as mentioned above, the Hungry Agents suddenly stop which, in an informal sense, looks unnatural given that they are not dead agents. Answering this question correctly might indicate that the student conceptualizes the simulation exercise itself and might correlate to programming the simulation correctly in the future which would relate to Research Question 1.

Question 8.

Description: Question 8 is posed after students have programmed the Fox and Rabbit dying of hunger and decomposing, eventually turning into Grass Agents. It asks how realistic the decomposition in the simulation is.

Relationship To Research Questions: This question relates to Research Question 2 in that students need to compare what they know about decomposition to what they observe in the simulation. Being able to point out how the simulation is inaccurate compared to the real life science phenomena

might be one way to solidify the concept in their mind and heighten understanding.

Relationship To GVC: This question relates to three Science Standards of the GVC: Science Standard 5, Science Standard 3 and Science Standard 1. The reason it relates to Science Standard 5 is that it involves students thinking critically about the simulation itself and where the representational system might break down. In this case, the animals decompose fairly quickly even in the reduced scale of simulation time. The question is somewhat contrived in that students do not know the average decomposition time for an animal but do know that decomposition is a lengthy process. This also relates to Science Standard 1 and LS3-D because students are explaining why this is unrealistic and hopefully, linking to evidence that shows it is unrealistic (possibly from previous class lectures).

Additionally, this question touches on Science Standard 3 as well, specifically LS12-A, as this is the first point in the simulation that students see something that might limit the size of a population, namely a lack of food.

Phase 1: The correct answer would indicate that students catch the disconnect between the representational system of the simulation wherein the Fox and Rabbit Agents decompose immediately and the real world where it would take much longer.

Phase 2: If students actually program this simulation correctly they would conceivably see that the Rabbit and Fox Agents decompose quickly. The question is unfortunately phrased however, the student knows by reading it that the answer has something to do with the duration of decomposition.

Therefore, it is unclear whether the act of programming the simulation would yield the correct answer as it pertains to this question though it might.

Question 9.

Discussion: Question 9 occurs at the same point as Question 8 in the worksheet and asks students about why we might use the Dirt Agent to make all of our changes. The aim of this question was to get students thinking about the actual mechanics of the simulation they were programming. However, it is somewhat of an ill-defined concept. First off, some students might not get that Dirt Agent covers the whole level (keep in mind students were given this pre-made simulation world and did not construct any part of it themselves), and that the Dirt Agent only sees surrounding agents at the top of the stack regardless of where that Dirt Agent is located in its stack; the device for making this work is somewhat of an AgentCubes quirk that is not really made apparent to the students anywhere in simulation creation. In a certain sense, students could make the connection that the Dirt Agent layers the level or exists in most places of the level if not all and be able to answer the question without fully realizing how stacked agents work. However, being able to do that it is still not apparent what that specifically means. Students know that dirt does not really make Fox Agents go hungry so that reasoning seems trivial. The reason we may choose to do changes this way is because the Dirt Agent happens to be our background agent so it is ever-present in the world relating to the idea of anti-objects [30]. Given that students might not be able to legitimately answer this question as well as the other problems with it, this question will not be used in the data analysis.

Question 10.

Discussion: Question 10 occurs after the students have programmed the Hungry Fox Agent tracking the Rabbit Agent and the Hungry Rabbit Agent tracking the Grass Agent. It asks the students what do the Hungry Fox and Hungry Rabbit still need to do in our simulation. It further asks what patterns you would use to accomplish this.

Relationship To Research Questions: This question relates to both Research Question 1 and Research Question 2. To answer the first part of the question the student must understand that something is not occurring when the Hungry Fox Agent actually gets to the Rabbit Agent and when the Hungry Rabbit Agent gets to the Grass Agent. This is made apparent when the user runs the simulation and the agents travel to their targets and then nothing happens. In fact, the agent will eventually die in this case. However, it is hard to make the argument that the students would not understand that these agents should eat at this point.

The follow up question relates to Research Question 1 in that students have to abstract the eating interaction and map it into the domain of the Computational Thinking Patterns in order to predict a pattern that would work for this purpose. Given that students are provided with step by step instruction on how to create the simulation, questions like this are important in gauging if students actually follow the concepts needed to implement simulations using the Simulation Creation Toolkit.

Relationship To GVC: These questions relate to Science Standard 5, Science Standard 3, and Science Standard 1. The first question relates to Science Standard 3 because it deals with "How living things interact with each other

and their environment [56].” Specifically, as the simulation is currently constructed, the hungry agents are acting unrealistically in relation to their food source. This question tries to ask the student what the correct interaction with the environment might be in this situation. Similarly it relates to Science Standard 1 and LS2-A because students have to evaluate this observance and form a logical conclusion as to what should happen in the context of this simulation.

Science Standard 5 relates to the follow up question as it asks students what they might do to model this behavior in this context. In this way they are actually acting out, on a small scale, how a scientist might create an aspect of a representational system of some real world phenomenon.

Phase 1: If students say something to the effect of the Fox Agent needs to eat the Rabbit Agent and the Rabbit agent needs to eat the Grass Agent they understand what exactly needs to be implemented after the tracking aspect of the simulation is complete. If students answer a valid pattern for the follow up question then they understand how to possibly implement the eating interaction. A valid pattern could be the Change Pattern or the Absorb Pattern.

Phase 2: Running the simulation might give students insight into this question as the Hungry Rabbit Agent, for example, would get to the Grass Agent and then just wait around until it died. So in this sense, a student might realize that the Hungry Rabbit Agent has to eat the Grass Agent somehow. However, this is not the only way a student can come to this conclusion, and it is impossible to know after the fact whether a student ran

this simulation to get this answer or not or if it was some sort of combination of running the simulation and prior knowledge. As mentioned above, it does not seem sensible to think that a student would be clueless as to what the Hungry Rabbit Agent should do when that Agent arrives at the Grass Agent.

Question 11.

Discussion: Question 11 occurs after the student programs the Fox Agent to eat the Rabbit Agent. The question asks if the Hungry Fox Agent is programmed to eat all the Rabbit Agents, how might this be unrealistic, and furthermore, what changes might we make to create a more realistic simulation? The basic idea behind this is for students to understand a logic error in the code they are creating—namely the Fox Agent eating an already dead and possibly eaten Rabbit Agent. This question is ill phrased in that it is not entirely unrealistic that a Fox Agent eats a Dead Rabbit Agent or even two Fox Agents share a Dead Rabbit Agent. There could be many correct answers here; this question will be looked at for the results because it might be interesting to see how students happened to answer this ambiguous question, but how it relates to the Research Questions and the Guaranteed Viable Curriculum as well as Phase 1 and Phase 2 is hard to determine without knowing how students interpreted this question.

Question 12.

Discussion: Question 12 takes place after the students implement the Grass Agents changing the Hungry Rabbit Agents back into regular Rabbit Agents and asks the students to run the simulation to find out how many seconds it takes for all the Fox Agents and Rabbit Agents to die out. It should be noted that at this point it is hypothetically possible to tweak the parameters such

that the system stays in equilibrium, however given the initial values placed in the specifications for the patterns implemented, this does not occur.

Relationship To Research Questions: This relates to Research Question 2 as it in part motivates the idea that mating needs to be added to the simulation to make the Predator/Prey model more accurate. In a sense, this question along with the simulation students have programmed up to this point has the potential to enhance understanding as it could motivate this idea as students see the population of these agents decline to nothing.

Relationship To GVC: This question relates to Science Standard 1 and Science Standard 5. It relates to Science Standard 1, specifically LS2-A because students are recording data. It relates to Science Standard 5 and LS15 because students are gaining a greater understanding for the conceptual model they have built thus far; essentially they are recognizing that in the current state of the model, all the agents will eventually die off.

Phase 1: There is no way to determine a correct answer for this because every Agent Pattern has multiple percent chances associated with its specifications.

Phase 2: If the student programmed the simulation correctly up to this point, all the Fox and Rabbit Agents die off. If this occurs, there is no real correct answer, and it would be difficult to interpret what a given answer might mean as compared to another answer.

Question 13.

Discussion: Question 13 occurs at the same point as Question 12 and asks the students what pattern they would use to add mating to this simulation. This question, similar to Question 10, is meant to have students think about the patterns and how they might implement something they have not already implemented.

Relationship to Research Questions: This question relates to Research Question 1 in that if students are able to correctly pick or describe the pattern they need to use it means that they are on the way to correctly implementing the pattern, and using the Simulation Creation Toolkit to implement various interactions. It also marginally relates to Research Question 2 in that it reinforces to students that mating still needs to be added to this simulation, before they can answer this question possibly ,increasing their understanding of the Predator/Prey interactions and factors that grow populations.

Relationship To GVC: This question relates to Science Standard 1, Science Standard 3, and Science Standard 5. In Science Standard 1 LS3-A students have to use evaluate observations to formulate logical conclusions. In this case the student has to evaluate the possible patterns and find the one that best fits with the real world representation they are trying to model. In Science Standard 3 students must gain insight into the way living things interact with themselves and the environment. Conceptualizing the addition of mating in order to make the simulation more realistic gives students the opportunity to ponder how these agents in the model and the animals they represent in the natural world interact in terms of mating. This question also

generally relates to Science Standard 5 as it forces students to represent the phenomena of mating in the context of their simulation.

Phase 1: The correct answer to this question would be the Generate Pattern or a description of the Generate Pattern.

Phase 2: The process of simulation creation could give students an idea of what pattern to use by exposing them to the patterns via the pattern picker. Furthermore, noticing that the agents are not mating might motivate this question. However it is doubtful that this indicates that the student learned anything by specifically running the simulation they have created thus far.

Question 14.

Discussion: Question 14 occurs after the students have implemented mating using the Generation Pattern. It asks the student to record how long it takes for the first animal (Fox Agent or Rabbit Agent) to totally die off. Therefore, the time recorded should be when all the instances of the Fox Agent or Rabbit Agent no longer exist in the world. It should be noted that a better strategy for this question would be for the students to run the simulation a number of times and take an average time for the Fox or Rabbit Agents to completely die off, but given time constraints, we decided to have students just do one trial.

Relationship To Research Questions: This question does not really relate to any of the research questions; it is meant to set students up for subsequent questions.

Relationship To The GVC: This question relates heavily to Science Standard 1 in that students are starting to use the technology of the simulation they have created to record results; this acts as a control for manipulations students will later make.

Phase 1: There is no real correct answer to this question. It provides a basis for further experimentation.

Phase 2: Given no correct answer and the nature of this question, it is hard to say how simulation creation helped garner a deeper understanding of this topic.

Question 15,16.

Discussion: Question 15 asks students to adjust the percent chance parameter for Mating Rate, Hunger Rate, or Death Rate to try to increase the time before the first species (Fox Agents or Rabbit Agents) die off. Question 16 asks students to describe what they decided to alter.

Relationship To Research Questions: These questions relate to Research Question 1 and Research Question 2. If students understand that this concept of manipulating parameters in their simulation as an experiment, and are able to make the connection that these parameter manipulations are changes in pattern specifications in the Simulation Creation Toolkit, then it adds to the idea that this strategy for integrating simulations into the classroom is promising relating to Research Question 1. It relates to Research Question 2 because by manipulating these parameters students can see how the simulation itself changes giving students the opportunity to realize that

these rates in the simulation have a direct effect on simulation populations as do these rates in real world Predator/Prey ecosystems.

Relationship to GVC: These questions relate generally to Science Standard 5 as students are manipulating the parameters of a model they previously created. It relates to LS15-A in the way that students can see how easy it is to manipulate and experiment on their model as compared to trying to run an experiment on this in the real world.

Phase 1: As long as the student manipulated a parameter this question is correct.

Phase 2: This question is setting up future understanding questions; by itself it does not say anything about student understanding of the underlying science concepts.

Question 17.

Discussion: Question 17 asks why the student thinks changing the parameter they changed would increase the time for the Rabbits or the Foxes to die off. In a sense, the students are hypothesizing that the change in parameter they made to the system will prolong the time it takes for the last agent to die off.

Relation To Research Questions: This relates to Research Question 2 as the student is forced to think about the system parameters of this simulation and how those system parameters can be manipulated to change the outcome of the simulation. These system parameters, which are pattern specifications, are similar to the ways one would talk about actual ecosystems—i.e. at what

rate does an animal mate or how susceptible is a given animal to death. It also relates to Research Question 1 in the sense that if students are able to manipulate parameters correctly and then make predictions on what these manipulations accomplish, they will be able test multiple hypothesis quickly using the Simulation Creation Toolkit. This is a huge advantage of the Simulation Creation Toolkit and creating models to run experiments in-general.

Relation To GVC: This question relates strongly to Science Standard 1 and specifically LS3-C wherein students make predictions based on experimental data. In this case students are observing which agent dies off first and then changes a parameter such that they prolong the life of that agent. It also applies to LS3-B because students are creating a hypothesis and supporting the hypothesis with what they have previously seen in the simulation. This question also relates to Science Standard 3 LS12-A, as students have to identify what parameter values might effect the size of a given population and alter those to enable the population to last longer.

Phase 1: There are multiple correct answer to this; the answer is deemed correct as long as the student can justify that the pattern specification they are altering should lead to the agent surviving longer.

Phase 2: Running the simulation does help students understand that one agent is eliminated before other agents. It is hard to argue that the student would have been able to answer this question before creating the simulation, especially justifying which agent's parameters to alter. This question can also be interpreted as a deeper understanding of how this system works and in

turn a deeper understanding of the system it is representing. This looks like it could be a Signature Phenomena candidate because the student must be able to reason out a change to simulation to cause a certain effect given the simulation run up to this point.

Question 18,19.

Discussion: Question 18 and 19 ask students to run the simulation with their parameter change, record what happened, and state why they think they got this result.

Relationship to Research Questions: This relates to Research Question 2 again because students are using the simulation to reason out a deeper understanding of what effect their manipulation had on the simulation. In turn, they are gaining insight into what these percentages, such as mating rates or death rates for a given animal, actually mean in terms of a real world ecosystem.

Relationship To GVC: These questions link to Science Standard 1 LS3-A, LS3-B, and LS3-D. It relates to LS3-A and LS3-B because students are evaluating data to form a logical conclusion as they try to explain what happened in their particular trial; it relates to LS3-B because students are using the evidence of their simulation run (albeit limited) to state if their hypothesis is supported or not. These questions also relate to Science Standard 5 LS15 in that students are using the model they created for prediction as to what would happen when they vary the parameters and explanation of why the simulation run worked out a certain way.

Phase 1: There are many correct answers to this simulation. As long as the student reports what happened and gives a reasonable response as to why it may have happened, the answer should be deemed correct.

Phase 2: It could be that if a student creates a simulation incorrectly, they do all the steps wherein they change a parameter, predict what is going to happen etc. and then reach an incorrect solution. Even in this case students are exposed to changing a parameter to test the model. Given that the student created the simulation correctly, one might expect that this question would be unanswerable without the simulation. If students are able to get a reasonable conclusion from this question it might indicate a deeper understanding of the topic they would have not had before creating the simulation.

Question 20.

Discussion: This question asks students to try a few more values and asks what got them the most time before their agents died off (if they died off at all). Given the time constraints, this question was included to allow students some open ended exploring on their own before they proceeded on past this experimental section. In an ideal world, students would have created their simulation and then spent maybe one or two class periods experimenting on it; unfortunately, this was not possible. Therefore, this was a way to allow students to "play around" with their simulation they had just worked on. These answers will be interesting to look at to see what students tried but this question is more an exploratory question rather than a question that can assess student understanding.

Question 21.

Discussion: Question 21 defines system equilibrium as every predator and prey that dies is replaced leaving their populations more or less the same. In class students were asked to play around with the system and if they could get the system into equilibrium, and then, were asked how hard is this simulation to keep in equilibrium. It should be noted that, though possibly related, this question is not referring specifically to energy equilibrium; just equilibrium among the Fox and Rabbit Agent populations.

Relationship To Research Questions: This question relates to Research Question 1 in that, as mentioned previously, if students are able to manipulate parameters in an attempt to get a desired effect it implies that students will be able to run multiple experiments by manipulating parameters. This question relates to Research Question 2 as it introduces the concept of system equilibrium and asks students to try to put their system into equilibrium by changing parameters. This exercise has a chance to give students better hands on understanding of system equilibrium and the difficulties in tuning parameters to keep a system equilibrium.

Relationship To GVC: This question relates to Science Standard 1 LS3-A as students have to interpret their observations of the simulation runs post parameter changes to come to a logical conclusion regarding keeping this simulation in equilibrium.

Phase 1: The correct answer to this question would indicate that it is hard to tweak these system parameters to get this simulation into equilibrium.

Phase 2: This question is a complicated one to analyze. In one sense a user could put the system into equilibrium by changing the mating rate of certain agents to something abnormally high. In that scenario, the agents would mate before they die of starvation regardless of food supply. In such a case, the user might think that the system was easy to get into equilibrium though the method of achieving equilibrium is unrealistic. Given the way the simulation is constructed this question could easily lead the student into a misconception as easily as it could a better understanding of system equilibrium. Therefore it is hard to say if the simulation actually increases student understanding on the idea of equilibrium.

Question 22, 23, 24.

Discussion: The worksheet gives students an example biomass graphical representation of trophic levels in an ecosystem. Then these questions ask students to calculate the biomass in the initial condition of the simulation for both the Fox Agent and the Rabbit Agent given that the Fox weighs 15 pounds and the Rabbit weighs about 4 pounds. Remember from Question 1, that the simulation world has 36 Fox Agents, 54 Rabbit Agents, and 89 Grass Agents. Finally it asks the student if our simulation is realistic and asks how they might change the simulation if it is not.

Relationship To Research Questions: This relates to Research Question 2 in that students are using the simulation as a prompt to further their understanding of biomass at various trophic levels.

Relationship to GVC: This relates to Science Standard 1 because students are interpreting the initial condition of the simulation itself as compared to how

an actual ecosystem would be structured and hopefully forming a logical conclusion about how to change the simulation to make it more realistic. Similarly, since students are looking at making the model a more realistic representation of the world, these questions also relate to Science Standard 5.

Phase 1: A correct answer would indicate that students understand that the Rabbit population cannot sustain the Fox population in the initial conditions of the simulation; thus, the student should say something like increase the number of initial Rabbit Agents or decrease the number of initial Fox Agents.

Phase 2: This question can be answered without students ever programming the simulation, as it deals with initial conditions of the simulation world provided to students, so it cannot really be said that the practice of programming the simulation increases student understanding as it pertains to this question.

Question 25.

Discussion: Finally, students are given little guidance and asked to make a Poacher Agent that shoots Bullet Agents from scratch. The Poacher Agent's Bullets should only affect the Fox Agent population. The question is then posed does the introduction of a Poacher Agent make the system easier or harder to keep at equilibrium. This question is not as straightforward as initially thought; for example, if students have unrealistic parameters an introduction of one or many Poacher agents could help keep the system in equilibrium (i.e. if students have Fox Agent mating rates extremely high or death rates extremely low).

Relationship To Research Questions: This is the first time students are asked to create their own agents and pick the correct patterns to implement without any guidance. One aim of this was to allow students to have the opportunity to create their own agent after using these premade agents the whole simulation. At this point of the unit, the student is almost finished and so they can spend as much or as little time as they would like creating their agent. This relates to Research Question 1 because students probably use to create the pattern to do something on their own; the scaffolding of the tutorial is removed and it is up to the student to figure out how to create this using the Simulation Creation Toolkit. Furthermore, students may patterns they have not up to this point—namely the Absorb Pattern and the Directional Movement Pattern. It also relates to Research Question 2 because we are introducing Poaching into the simulation hopefully getting the idea across that human interactions also have the ability to effect these populations.

Relationship to GVC: This relates directly to Science Standard 3 LS12-A, regarding the types of factors that can limit a population, and LS12-B, regarding the impact humans have on the environment and how that effects the survival of populations and entire species. Unless unrealisitic parameters are used, students should see that the Poacher Agent has a non-beneficial impact on the Fox Agent population.

Phase 1: Given realistic parameters, the student should come to the conclusion that it the system equilibrium is changed with the introduction of the Poacher Agents knocking off the Fox Agents. An answer to this effect would be considered correct. If the student has unrealistic parameters then

the Poacher Agent might actually help keep the system in equilibrium, and in this case, that answer would be considered correct.

Phase 2: The act of simulation programming may or may not enhance student understanding based on whether the parameters values are correct or not. In fact, if the values are incorrect the student may get an incorrect understanding of the role Poacher Agents play in the simulation and thus, might have an incorrect understanding of the real world phenomenon (in this case poaching). However, if the student is using realistic parameters, the student should gather that affects the system equilibrium.

4.2.4 Discussion Of Predator/Prey Unit

The Predator/Prey unit is an attempt to answer the Research Questions of this thesis while also trying to integrate the exercise into the classroom curriculum as seamlessly as possible. In retrospect, there are many things that could have been done differently in both the design and execution of the Predator/Prey unit, however multiple constraints made these options infeasible.

To get the maximum benefit from the Simulation Creation Toolkit, ideally, students would use the tool over the course of the semester or year. In this scenario, the overhead of teaching students the Simulation Creation Toolkit, over a few days, would pay off in that students could build multiple simulations with numerous experiments attached to each simulation the students create. In this way students not only create the simulation in a relatively easy manner, but, using the specifications of each pattern, can carry out multiple experiments extremely quickly. In this scenario, Research

Question 1 could be easily answered by analyzing how well students were able to create simulations over the whole curriculum.

Instead of taking time for students to learn the system, the tutorial/worksheet for the unit had to integrate not only how to use the Simulation Creation Toolkit, but also, how to create the Predator/Prey simulation. Furthermore, the steps students take to create the simulation are made extremely explicit to the extent where they are told where to click at various screens. Though not exactly what was intended for this system, this strategy is very similar to previously used introductory AgentSheets units. For example, the Frogger, Pacman, Space Invaders, and Sokoban tutorials on the Scalable Game Design wiki also state each step a user must take very explicitly as well; this strategy is not just limited to games-- tutorials of simulations such as the Contagion tutorial also give students step by step instructions on how to complete the simulation.[8]

Providing the students with step-by-step instructions has its own associated advantages and disadvantages in the context of using the Simulation Creation Toolkit in the classroom environment. The main disadvantage is that it compromises one of the potential main benefits behind the Simulation Creation Toolkit, namely that it makes simulations easier to create as students are able to implement patterns by programming by analogy abstracting out agents and preserving interactions. There are also a few advantages, however, to this strategy. With limited time, this method allows the instructor to lecture for a small duration leaving the rest of the class for students to create their simulations. Students in this study may have never used AgentSheets or AgentCubes; for these students the whole

[8]http://scalablegamedesign.cs.colorado.edu/wiki/Scalable_Game_Design_wiki

construct of agents, depictions, behaviors, and worlds is brand new. Trying to teach a 4 day unit wherein not only the Simulaton Creation Toolkit system must be explained, but also, the basics behind AgentCubes, and then expecting middle school students to be able to create a complicated simulation from beginning to end using this system they have just been introduced to without scaffolding seems unrealistic. Recall that in a prior experience, high school students who had had a week of AgentSheets took over a week to create a much more basic version of the Predator/Prey simulation (see section 1.3). Finally, in middle school classrooms there are often English as a second language students present as well as learning disabled students. Having each step graphically represented could allow these students to participate in the act of simulation creation without being left behind because something is not entirely clear, and there is not enough time to explain all the concepts covered in this unit in depth and in a variety of ways.

In lieu of having students use the Simulation Creation Toolkit from scratch with little scaffolding, the worksheet instead asks questions as to how students might implement certain interactions (mating, eating etc.) before the implementation of those interactions are described. Furthermore, students were told that any answer they put down is fine and were told not to go back and change their previous answers. This allows us to get a glimpse into how students would have employed the patterns present in the Simulation Creation Toolkit with no scaffolding provided, without having students get lost in the unit. Future research should involve a larger scale study wherein students actually use the Simulation Creation Toolkit over the whole semester or year to assess the full advantages of the system.

The study itself could have been implemented more effectively. To answer Research Question 2 it would have been ideal to hand out a pre-test to students to identify where misconceptions about Predator/Prey interactions might have still been present. Then, at the end of the unit, the students could have taken a post-test with the pre and post test results being compared to see if the act of simulation creation and experimentation actually removed these misconceptions. There are a few reasons this did not occur. The constraints on both in-class time and the time it took to create the system for the class as well as coordinating all the aspects of the project (IRB with the University of Colorado and the Boulder Valley School District, reserving the lab, etc.) made this strategy infeasible. Future research should incorporate this or a similar idea.

As a possible replacement for this strategy, after the worksheets were handed in but before looking at the students' answers, there was an attempt to identify questions that could only be answered by programming the simulation. These types of questions, as stated above, are said to identify Signature Phenomena. However, as shown in the Phase 1 and Phase 2 analysis of the worksheet questions in section 4.2.3, these types of questions are not easy to find. In fact, in the original research that employed this strategy, no questions exhibited Signature Phenomena [57]. In this worksheet, only a few questions out of 25 had the potential to exhibit Signature Phenomena. Even then, it is hard to say whether answering these question actually shows that the act of simulation programming enhanced understanding of the topic.

This thesis is an initial investigation or exploration in using Computational Thinking Patterns to create simulations. The aim of this study is to answer whether this is a good avenue of further investigation and

an effective strategy to pursue for integrating simulations in the classroom. The study itself is aimed at having students use the system and begin to gain insight into the Research Questions. Though the system is not used in an ideal manner, we can still begin to answer Research Question 1 with the data obtained. For one, if students are not able to create simulations given step-by-step instructions, then there is little chance they will be able to create simulations if such scaffolding is removed. Furthermore, by analyzing questions that have students predict the possible pattern they should use to create a given interaction as well as analyzing how they implemented the Poacher Agent, provides insight into how successful the system might be once the scaffolding is removed. Similarly, though Research Question 2 is tough to answer, we can still obtain evidence as to whether the Simulation Creation Toolkit strategy is promising in terms of increasing student understanding. In a simple sense, the use of this simulation creation exercise enabled questions that Marco Conachionne thought touched upon often-misunderstood topics. These topics were placed into the context of the simulation the students were creating; thus, even with the lack of Signature Phenomena the simulation at the very least enabled the unit to target these topics. Given that this thesis is an initial investigation, more research must be preformed; these subsequent studies should incorporate the above ideas to better answer the Research Questions as well as other questions regarding programming simulations at the Computational Thinking Pattern level.

4.3 Study Implementation

The study for this Predator/Prey unit took place at Centenary Middle School. As mentioned above, two teachers were involved. Marco Cornacine

teaches 7th grade Life Science and the unit was tailored to his class. Marks Savs teaches the 6th grade Computer Class. This section will briefly review the student populations and talk about what occurred in the classroom. The IRB specifics and the specific numbers of student data taken will be outlined in Chapter 5.

4.3.1 Classroom Populations

Two groups of students took part in this study: 7th grade Life Science students and 6th grade Computer Class students.

The 7th grade Life Science classes consisted of 4 periods: 2, 3, 5 and 7. This study took place over 4 days from April 16th to April 19th. This 7th grade population had worked with simulations before but most had never created their own simulation. Furthermore, most of these students had prior AgentSheets experience (in 6th grade) but had never made an AgentCubes game. The Predator/Prey topic was something these 7th grade students had just covered, so they were getting this unit in-context.

The 6th grade Computer classes consisted of 2 periods: 5 and 6. This study took place over 3 days: April 26th, April 27th, and April 30th (with a weekend in between). These students, for the most part, had minimal experience with AgentCubes and AgentSheets. Furthermore, they had minimal to no experience with simulations before. Finally, since they were not studying ecosystems like the 7th grade class, the 6th graders got this unit out of context. As mentioned above, this group was included in the study to see how effective the tool was for students who had never done any AgentSheets/AgentCubes game creation.

4.3.2 Class Structure

This section will outline the structure of each class. Both unit studies took place at the Macintosh computer lab at Centenary Middle School. On the first day the instructor and I spent 15-20 minutes introducing the unit. This included the idea of simulations and modeling, the Predator/Prey model we would be building, and introducing students to the agents we would be using in our simulations.

At this point I had the students participate in a role-play exercise wherein students were given the role of the Rabbit Agent, the Fox Agent, and the Grass Agent. The students would roll a die to decide which direction they would move. They would also roll a die to see if they became hungry or not. When the Rabbit became hungry she/he would start to move towards the Grass Agent. When the Fox became hungry she/he would move towards the Rabbit Agent. The hungry agents would roll after each move to see if they died of hunger. If the Hungry Fox Agent got to the Rabbit Agent, the Rabbit agent would change into a Dead Rabbit, changing the Hungry Fox Agent back into a regular Fox Agent, and similarly, if the Hungry Rabbit Agent got to the Grass Agent, the Grass Agent would change into the Eaten Grass Agent and the Hungry Rabbit would change back into a regular Rabbit Agent. This was a way to show students what they would be creating in this unit.

After the role play exercise, I would introduce the concept of patterns and the Simulation Creation Toolkit, with a few examples, and answer any questions students may have had. Finally, the tutorial/worksheet was handed out and students started working. Subsequent days started with a 5-10 minute lecture reviewing what students had accomplished thus far, clearing up any common confusion among students, and looking forward to

what the students were to do that day. The class instructor would also emphasize concepts that he wanted the students to understand. After that, students would get to work with the instructor and I going around and helping students when they raised their hands. At the end of the day I would collect all the computer data and worksheets from the students and return it to them the next day so they could continue their work.

Overall the data collection went smoothly. Scanned 1 page hand-written teacher diaries for each day can be found in Appendix D. Some of the ideas written about in the teacher diary will be discussion points for when the results are presented in Chapter 5.

4.4 Analogical Reasoning Study Implementation

The analogical reasoning study was added to the thesis in order to see if users could create simulations using the Simulation Creation Toolkit with a brief introduction to the toolkit and a general description of what should be created. Given the step by step instructions provided to students in the classroom, this study was a way to see how users might respond to the system given less scaffolding. It also provided an opportunity to see how effective the system is over a few different game and simulation creation activities rather than just one. Specifically, through this study we can better analyze which patterns were intuitive for implementing particular interactions and which patterns were not. The basic question of this study was can users successfully reason out which patterns to implement given high level English descriptions of a particular game or simulation.

To this end, study participants were given a brief (~5-10 minute) introduction to the Simulation Creation Toolkit and AgentCubes.

Participants were then provided with three programs: A Pacman game, an Epidemiology simulation, and a Predator/Prey simulation. All the programs provided the participants with every agent and a premade world they would use; the only thing missing from every program were the agent behaviors. The participants would use the Simulation Creation Toolkit along with the brief program description provided and attempt to create the description as best they could. As they worked through each part of the description they would talk aloud as to what they were trying to do. I would sit behind them taking notes. I would only talk if the user asked me a question directly or if the user encountered a bug of the system; I would only answer the question if it was truly something the participant was not afforded the information to figure it out on their own. The participants worked on the programs as long as they wanted to and they would tell me when they reached a stopping point at which they were satisfied with their creation.

After the participants were finished with all three programs they answered two questions. The first question asked what the participants would change or modify about the tool. The second question asked what pattern(s) in what simulation(s) did the participants have trouble implementing, if any, and why.

Six people participated in the study. As mentioned above, almost all of them (except one) were college students ages 20-33. Three of the participants had previously attended a Scalable Game Design Summer Institute at the University Of Colorado Boulder. Therefore, these students had seen AgentSheets and AgentCubes before as well as had been introduced to the concept of Computational Thinking Patterns. The other three participants had never seen AgentSheets/AgentCubes before and had never heard of Computational Thinking Patterns before this study.

The following sections will outline each study. The materials provided to the users for the study can be found in Appendix E including the pictures of agents and worlds provided for each programming exercise in Appendix E.2.

4.4.1 Pacman Game

The first program participants were asked to program was Pacman. It was decided to give this as the first exercise because most people have played Pacman and it is relatively easy to create (i.e. small number of interactions). The agents in the game consisted of Pacman, a Ghost with four depictions, a black Background, Pellets, and a Wall agent. The following is the description that the participants were asked to follow as best they could.

"Pacman moves with keyboard keys. All the Ghosts pursue Pacman and when they get to Pacman, Pacman disappears. Pacman eats pellets as he navigates around the level. Neither Pacman nor the Ghosts can go through the blue walls."

Unlike the previous study, since users can implement this description any way they see fit, and since there are multiple ways to implement this description, it makes more sense to assess these simulations with respect to how many interactions students were able to implement correctly. The following four interactions correspond to the above description.

1) Pacman moves with keyboard keys and cannot go through Walls as he moves
2) All Ghosts pursue Pacman and do not go through Walls as they move.
3) When the Ghosts get to Pacman, Pacman diseappears.
4) Pacman eats pellets as he navigates around the level.

4.4.2 Epidemiology Simulation

The second program participants were asked to create was a simulation of disease spreading among a population. Like the Pacman game participants programmed previously, the Epidemiology simulation has only four interactions. The agents in this simulation are a Background Agent and a Person Agent with 2 depictions: Sick and (normal) Person. The following is the description participants were given.

"All depictions of the person move around randomly. A healthy person has a 30% of becoming sick each second that healthy person is next to a sick person. A sick person has a 30% chance of recovery every second. A sick person also has a 10% of death (i.e. disappearing) every second."

The following four interactions correspond to the above description and were used to assess the programs participants created.

1) All depictions of Person agent move around randomly
2) Healthy Person has a 30% chance of being sick if next to a Sick Person
3) A Sick Person Agent has a 30% chance of recovery every second.
4) A Sick Person also has a 10% chance of death every second.

4.4.3 Predator/Prey Simulation

The Predator/Prey simulation was the final program participants created. This simulation was a subset of the simulation Centenary Middle School students were guided through creating in class. This exercise was included in part to see how realistic it would have been to have a unit wherein middle school students created this simulation without guidance. The agents in the simulation are Dirt; a Fox Agent with 3 depictions: (normal) Fox, Hungry Fox, and Dead Fox; a Rabbit agent with 2 depictions: (normal) Rabbit and Dead Rabbit. This simulation is the most challenging of all the exercises because it necessitates the most interactions with some

being unintuitive. Also, unlike the previous Epidemiology simulation, the percentages and times of the different patterns are left up to the participants themselves. The following is the description participants were provided with.

"Foxes and Rabbits move randomly. Every so often the Fox Agent gets hungry; at this point the Hungry Fox tracks the Rabbit Agent. If the Hungry Fox gets to the Rabbit Agent, it kills it and is no longer hungry. Eventually the dead Rabbit decomposes. Hungry Foxes can also sometimes die of Hunger. Finally, Foxes sometimes reproduce with other Foxes creating a new Fox at some percentage; Rabbits sometimes mate with other Rabbits creating a new Rabbit at some percentage."

The following ten interactions correspond to the above description and were used to assess the programs participants created.

1) (normal) Fox moves randomly.
2) (normal) Rabbit moves randomly.
3) Fox becomes hungry.
4) Hungry Fox tracks the Rabbit.
5) If Hungry Fox gets to a Rabbit, it kills it.
6) Once a Hungry Fox has eaten a Rabbit, it is no longer hungry.
7) The Dead Rabbit decomposes.
8) The Hungry Foxes can sometimes die of hunger.
9) Foxes reproduce with other Foxes.
10) Rabbits reproduce with other Rabbits.

4.4.4 Analogical Reasoning Study Discussion

As mentioned above, the analogical reasoning study was a way to gain insight into if it is realistic to expect students to create simulations using the Simulation Creation Toolkit without heavy guidance. Ideally, this

study would be done with middle school students after they had completed their first Predator/Prey unit. However, given constraints for time and IRB approval (including BVSD district approval), it was decided that having students who are over 18 would be easier and begin to provide this initial insight. Future research should incorporate looking at to what extent actual middle school students are able to create simulations without guidance once they have been introduced to patterns and created an introductory simulation with step-by-step instructions.

Research Question 1 asks if students programming at the Computational Thinking Pattern level can successfully create simulations of scientific phenomena they are presented within class? There are two benefits of the system as presented that may facilitate users creating simulations. The first benefit is that patterns take less time to implement than creating behaviors using if/then conditionality statements. The second benefit is that users can hypothetically create simulations via analogy with interacticons provided by the simulation. Both benefits are in part explored by the Centenary Middle School study. Students attempt to create a more complicated version of a simulation that traditionally take a week with high school students, in four days. The second benefit is mitigated by providing students with step by step instructions on how to create the simulation. One way the Centenary Middle School study tries to touch upon the second benefit is by asking students what pattern they might use to create a given interaction. The analogical reasoning study is another way to gain initial insight into this second benefit. Table 7 summarizes the analogical reasoning study.

In lieu of this, three participants who took in the Summer Institute were included in this small-scale analogical reasoning study. These

Evaluation Objective	Outcome Measures	Data Sources	Method
RQ1	How many of the interactions were participants able to implement correctly? What types of interactions did participants have trouble implementing using the system and why?	Simulations from participants and notes from their simulation creation activity.	Participants create three programs: Pacman, Epidemiology, and Predator/Prey

Table 7: Summary Of Analogical Reasoning Study

participants had created simulations using AgentSheets before and been previously exposed to the idea of Computational Thinking Patterns; in this respect these participants were akin to a user who had an introduction to the concepts of the Simulation Creation Toolkit. Obviously, these participants had not seen or used this specific tool before and were much older than middle school students themselves. Therefore, it is difficult to say anything conclusively about whether the problems these participants encountered with the tool would be similar to the problems middle school students might encounter. What could be discovered are general areas where the Simulation Creation Toolkit needs to be modified. Furthermore, if these participants were not able to use the system successfully without guidance, it implies that the Simulation Creation Toolkit should be changed before trying to having middle school students do the same.

Three participants who had never seen AgentSheets/AgentCubes before were also included in this analogical reasoning study. These participants were included to see how realistic it was to have users who had

never seen an agent before created simulations using the system. Furthermore, including these participants gave insight into assumptions made in the Simulation Creation Toolkit that may not be valid for someone with no prior experience with AgentSheets/AgentCubes.

4.5 Study Summary

This thesis serves as an initial exploration into the Simulation Creation Toolkit. Two studies look at how the Simulation Creation Toolkit enables students to create simulations and whether this act using the Simulation Creation Toolkit actually increases student understanding. The first study is a Predator/Prey simulation unit at Centenary Middle School. This unit includes seventh grade students in Life Science and sixth grade students in the exploratory wheel computer class. Furthermore, a worksheet that students work through as they complete this study serve to give insight into both research questions.

The second study looks at how users employ the system when not provided with step-by-step guidance. This study looks at Research Question 1, and specifically, aims to give insight into the ability of users to create simulations purely by analogy using this system. Two types of users are analyzed: users with prior AgentSheets/AgentCubes experience that have been exposed to Computational Thinking Patterns before and users with no prior exposure to AgentSheets/AgentCubes.

The following chapter will look at the results of these studies and discuss what they mean and the factors that might have led to those results. Chapter 6 will look at what the results mean in terms of the thesis Research Questions.

CHAPTER 5

5. RESULTS

This chapter will present the results of the studies described in Chapter 4 and discuss what these results might indicate in general. First the middle school study will be presented and then the results of the analogical reasoning study. Any discussion of the results pertaining to the research questions will be reserved for Chapter 6.

Since middle school students are categorized as a "vulnerable population" by the University of Colorado's Institutional Review Board, the students, only data from students who returned the Parent Consent Form were included in the results. Furthermore, students who missed a day and/or did not turn in a project for every day or did not turn in the necessary worksheets are not included in the results.

The results include 66 students: 21 6th graders in Marks Savs's computer class and 45 7th graders in Marco Cornacine's Life Science class. The breakdown of 6th graders by class period is as follows: 10 students from the 5th period computer class and 11 students from the 6th period computer class. The breakdown of 7th graders by class period is as follows: 13 students from the 2nd period Life Science class, 12 students from the 3rd period Life Science class, 7 students from the 5th period Life Science class, and 13 students from the 7th period Life Science class.

The results are broken down into three main sections. The first section covers how well students were able to use the mechanics of the Simulation Creation Toolkit to program their simulation. The second section covers how students answered the various worksheet questions as they

worked through the simulation. The third section looks at how the worksheet question results relate to the simulation results and vice versa. Finally, the Analogical Reasoning Study results will be presented including the three programs attempted followed by a discussion of the study.

5.1 Results Of Simulation Creation Activity

These results look at how correctly students created the Predator/Prey simulation. Specifically, these results look at whether students were accurately able to create the patterns and corresponding specifications outlined in section 4.2.1. For each pattern there are two different ways of analyzing accuracy. The first method awards partial credit for a particular pattern. Namely, the simulation gets two points for the correct pattern and agents involved as well as one additional point for every specification the student correctly gets. The second method, referred to as "all or nothing" credit, awards 1 point for each pattern that is entirely correct; if any part of a pattern is wrong, that pattern gets 0 points. Since the experimental section of the worksheet necessitates students to change the percent chance specification for various patterns, no percent chance specification is taken into account. The following subsections review the total results and then splits the results by grade with a short discussion for each section.

5.1.1 Simulation Totals For All Students

The total data for all students includes 2 periods of 6th grade students from Marks Savs's class who had 3 days to complete the simulation and 4 periods of 7th grade students from Marco Cornacine's Life Science class who had 4 days to complete the simulation. Thus, these results use data from all 66 students involved in this study. The following radar charts depict the

partial credit scores totals for all classes combined. The data in this section only uses the simulations collected from the end of the final day (Thursday in the 7th grade class and Monday in the 6th grade class).

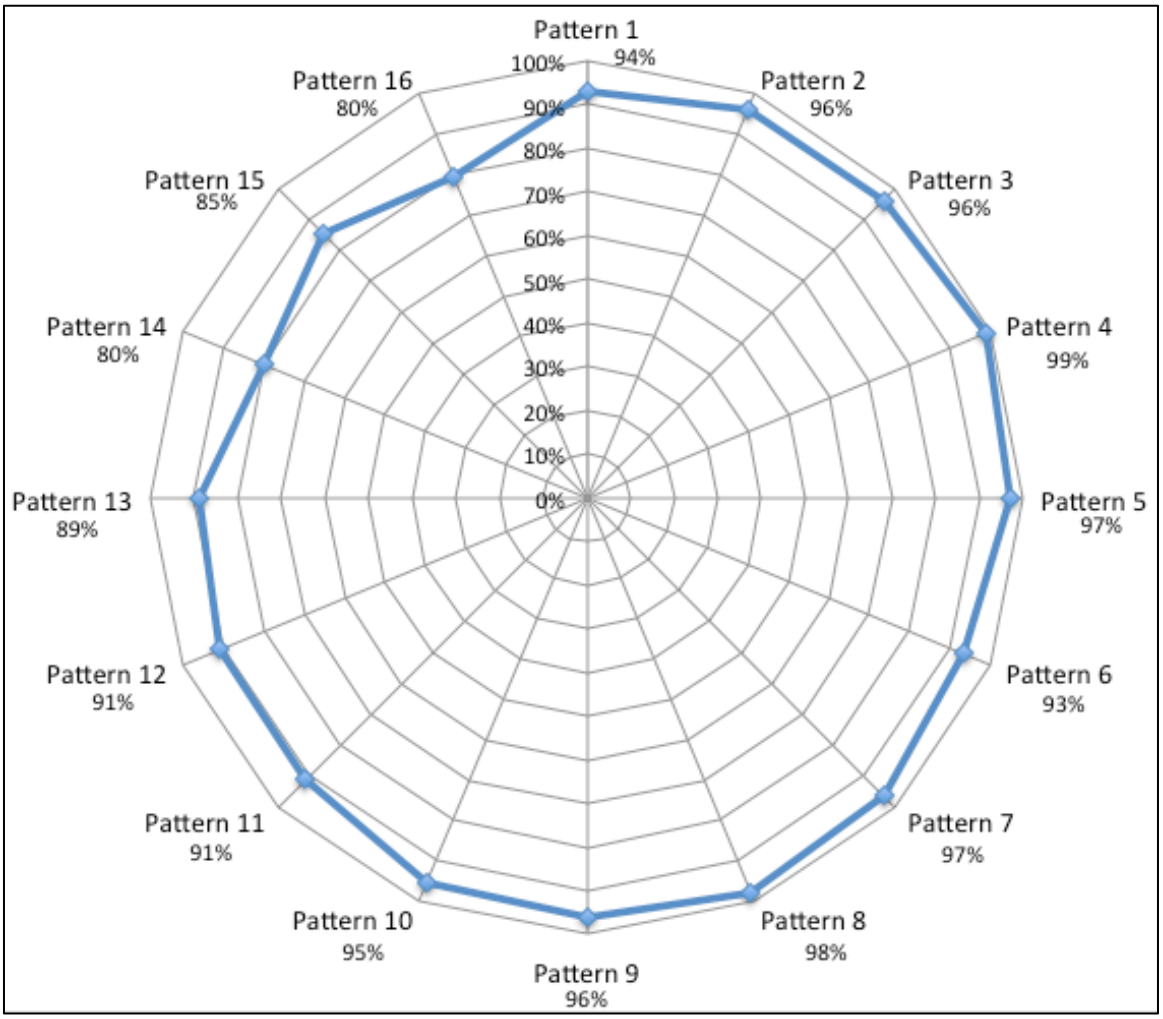


Figure 84: Final Partial Credit Pattern Averages For All Classes Combined

The radar chart in Figure 84 has 16 axes, one for each pattern students had to implement to correctly create the Predator/Prey simulation (see section 4.2.1). The values refer to the average amount students were able to get that particular pattern correct. For example, students in general were able to implement Pattern 9 95% correctly. Therefore, if every student implemented

every pattern correctly, Figure 84 would look like a 16-sided polygon with the blue line circling the perimeter of the graph representing each pattern at 100%.

Figure 84 combines two classes that had distinctly different experiences in creating the Predator/Prey simulation. The sixth grade class, for example, had 3 days to complete the simulation; As noted in section 4.2, it was estimated through discussions with both teachers that 3 days would equate to around Pattern 10. In contrast the seventh grade class had 4 days to complete the simulation. Furthermore in contrast to the seventh grade students who were participating in a Life Science class when they created this simulation, the sixth grade students were not studying ecosystems and thus received the unit out of context. Finally, most of the sixth graders had little to no experience with creating simulations and/or using AgentSheets/AgentCubes.

Even with these differences between classes, Figure 84 highly indicates that students were able to work the mechanics of the Simulation Creation Toolkit to create the Predator/Prey simulation. As expected, with the inclusion of the sixth grade class, the percentages start to drop off at the later patterns. The lowest percentage among the 16 patterns is Patterns 14 and 16 wherein students were on average 80% correct implementing this pattern. Pattern 16 is the last pattern students implement and therefore, the least amount of students would get to this particular pattern.

The following Figure depicts the all or nothing credit pattern averages for all the classes combined.

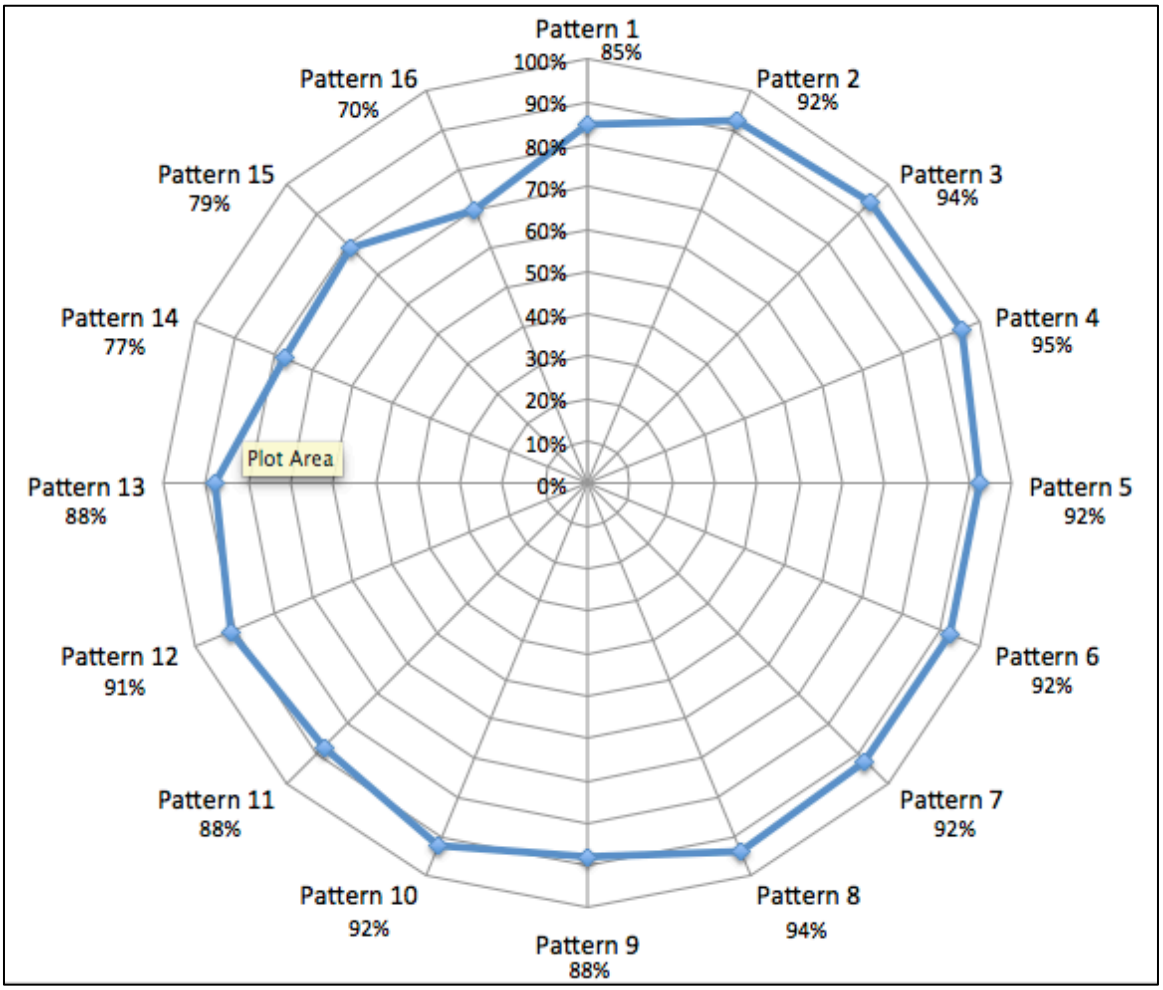


Figure 85: Final All Or Nothing Pattern Averages For All Classes Combined

Similar to Figure 84, Figure 85 has 16 axes with each axis representing different pattern that students had to implement for the Predator/Prey simulation. However, since there is no partial credit in the averages calculated in Figure 85, Figure 85 represents the percentage of students that correctly implemented a particular pattern.

This graph begins to show that students can indeed use the Simulation Creation Toolkit to create the Predator/Prey simulation. As with Figure 84, Pattern 16 has the lowest percentage associated with it at 70%. Therefore, at least 70% of students were able to get to Pattern 16 and implement it correctly.

The following table categorizes students into how many patterns they implemented correctly.

	Perfect Simulation	1 Incorrect Pattern	2 Incorrect Patterns	3 or More Incorrect Patterns
Number Of Students	23	11	11	21
% Of Students	35%	17%	17%	32%

Table 8: Categorizing Total Number Of Students By How Correctly They Implemented The Simulation

Table 8 shows that 35% of all the students were able to complete the simulation correctly. Furthermore, it shows that almost 70% of students completed the simulation with only 2 or less incorrect patterns. As with the above data, this includes the sixth grade class who had one less day to complete the simulation. With this in mind, let us now view the data submitted by each grade level.

5.1.2 Simulation Totals For The Seventh Grade Life Science Class

As mentioned before, the seventh grade Life Science class taught by Marco Cornacine, completed this unit over four days. The seventh grade class had prior exposure to AgentSheets as well as had studied ecosystems in class. Data was taken from 45 seventh grade Life Science students over four class periods. The following radar chart depicts the final partial credit averages for all seventh grade students.

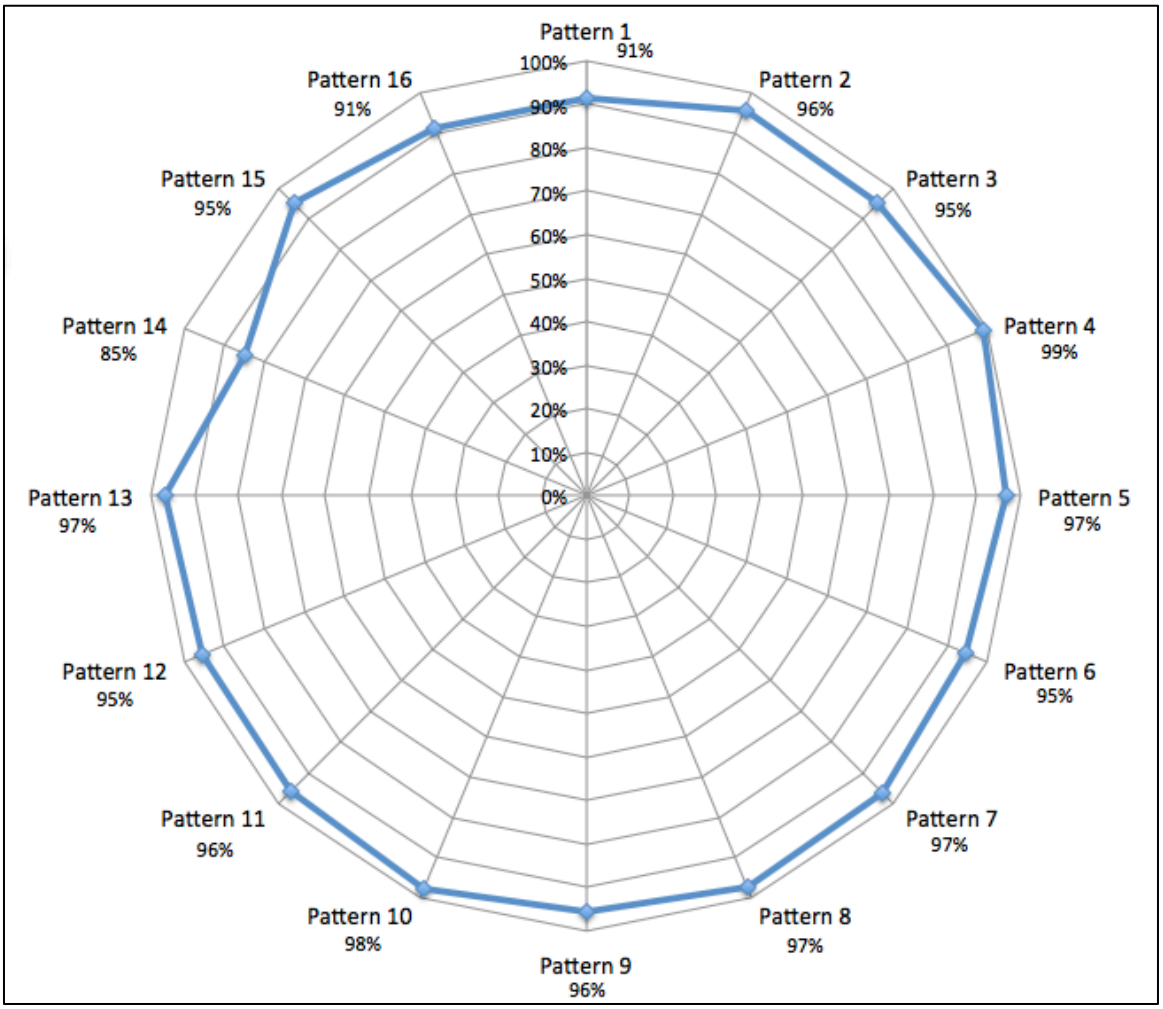


Figure 86: Final Partial Credit Pattern Averages For All Seventh Grade Students

Figure 86 indicates that the seventh grade students were able to create the patterns necessary to implement the Predator/Prey simulation. The patterns students had the most trouble with were Pattern 14, Pattern 16 and Pattern 1. Pattern 1 might have been troublesome because it is the first pattern students implement using the Simulation Creation Toolkit. Therefore, many students might have not completely understood how to use the tool at that point. Furthermore, a few of the classes were rushed on Monday, as mentioned in the teacher diaries, and with all the activities and new information presented to students on the first day, it could be easy to

overlook some of the pattern specifications. Pattern 1 and Pattern 2 have many specifications associated with them, which involve the 7 blocking agents for the Rabbit and Fox Agent random movement. Also, many students started to implement Pattern 1 on Monday, did not finish the implementation but moved onto implementing Pattern 2 on Tuesday. The simulation still works if a student does not specify all the blocking agents (it will just lead to Rabbit agents jumping on the Wall agent or other Rabbit and Fox agents); if a student does not realize that they didn't implement the pattern correctly they may not realize it by merely running the simulation.

Pattern 16, wherein the Rabbit agents reproduce, is the last Pattern implemented which means that the students are least likely to get to that particular pattern. The percentage associated with Pattern 14, wherein the Eaten Grass Agent changes the Hungry Rabbit into the regular Rabbit Agent, is somewhat anomalous. Pattern 14 is very similar to Pattern 13; one would expect the percentage associated with Pattern 14 would be akin to Pattern 13. The way that the percentage is calculated is that students can only get awarded partial credit for a pattern if the agents associated with the pattern are correct. Therefore, for Pattern 14 the Eaten Grass and the Hungry Rabbit Agent have to be selected in order for that particular student's simulation to be awarded points. However, some students selected the Regular Grass Agent instead of the Eaten Grass Agent. This would lead to an incorrect simulation because a Hungry Rabbit could possibly eat the Regular Grass Agent changing it into Eaten Grass and the Eaten Grass Agent would not change the Hungry Rabbit Agent back into a Regular Rabbit Agent. It should be noted that this error might not be easy to catch because if the Regular Grass Agent executes its rules first, it could also change the Hungry Rabbit Agent back into a Regular Rabbit Agent; at this point the

Regular Rabbit Agent would not change the Grass Agent into an Eaten Grass Agent. Therefore, the simulation would be incorrect but the student would see Hungry Rabbit Agents change back into Regular Rabbit Agents and see Grass Agents change into Eaten Grass Agents and so might believe that they programmed it correctly. Moreover, since students did not choose the agents associated with this pattern correctly, they would get zero points in both the partial credit analysis and the all or nothing credit analysis of their simulation. These zero scores for this pattern would skew the percentage lower. If this is in fact the case, we would expect that the all or nothing analysis percentage would be a little less than the percentage associated with the partial credit analysis (assuming that a few students would get the agents associated with the pattern correct but miss a specification). The following radar chart depicts the all or nothing credit pattern averages for all seventh grade classes.

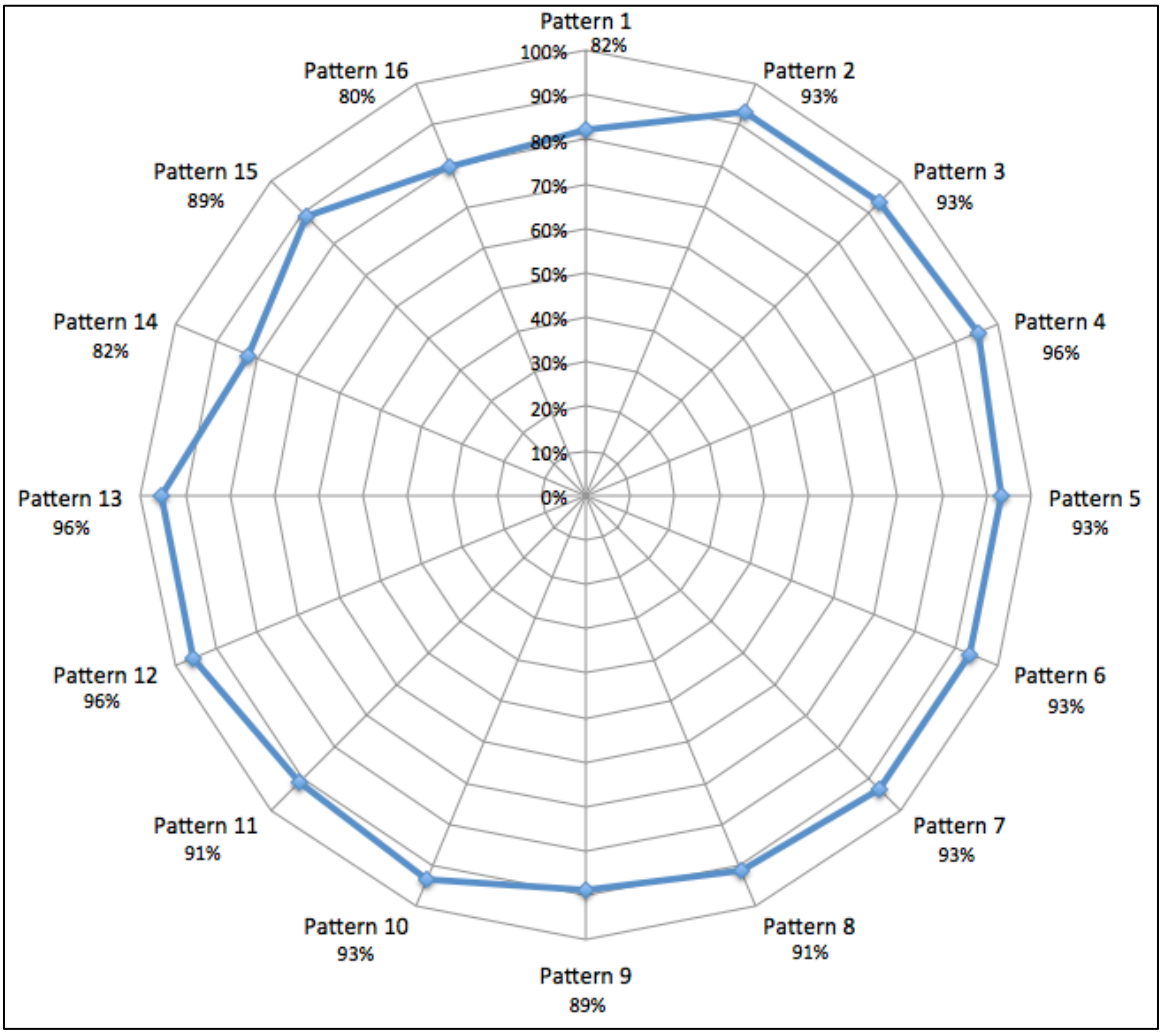


Figure 87: Final All Or Nothing Credit Pattern Averages For All Seventh Grade Students

Pattern 14, Pattern 16 and Pattern 1 in Figure 87 have the lowest percentage of students implementing them correctly. However the reasons seem different for each. For Pattern 14, the partial credit percentage and the all or nothing percentage are close at 82% and 85% respectively. This implies that, for the most part, the students who got Pattern 14 wrong usually got it completely wrong. As eluded to above, this shows that the students missed one of the agents involved in the pattern. An informal look at the simulations themselves corroborates this theory as many students had the Regular Grass instead of the Eaten Grass Agent selected for this pattern.

Pattern 16 and Pattern 1, on the other hand, seem to imply that students got the Agents correct for these patterns but not all the specifications correct. For example, Pattern 1 was, on average, implemented 91% correct by seventh grade students and only 82% of students implemented this pattern perfectly correct. Therefore, students tended to miss one of the specifications associated with this pattern. Given that this pattern had the most specifications to change in addition to this being the first pattern students implemented it makes sense that some students might miss a specification for this particular pattern. Similarly, the percentages associated with Pattern 16 drops off drastically from the partial credit analysis to the all or nothing analysis at 91% and 80% respectively. The nature of the Generate Pattern necessitates the need for many specification choices. For example, something can generate on a collision, with a certain percentage, or with a key press in any direction (see section 3.6.9). For this particular simulation both the Rabbit Agents and the Fox Agents created a new agent straight down with a certain percentage given that these agents saw another agent to the right. Since it was the last pattern, students may have only partly implemented it or could have easily missed one of the specifications because they were trying to get through all the worksheets. Furthermore, Pattern 15, also a Generate Pattern, declines from the Partial Credit percentages in Figure 86 at 95% to the all or nothing percentages in Figure 87 at 89%. Therefore, given the complexity of specification choices in the Generate Pattern, might allow this pattern, in general, to be more prone to student mistakes. This should be an avenue of further investigation, namely, how best to present the pattern specifications as not to confuse users.

The following table categorizes seventh grade students into how many patterns they implemented correctly.

	Perfect Simulation	1 Incorrect Pattern	2 Incorrect Patterns	3 or More Incorrect Patterns
Number Of Students	18	9	9	9
% Of Students	40%	20%	20%	20%

Table 9: Categorizing Seventh Grade Students By How Correctly They Implemented The Predator/Prey Simulation

Table 8 shows that 40% of seventh grade students had a perfect simulation. Furthermore, 60% of students had one or less mistake and 80% of students had 2 or less mistakes in their simulation. This data shows that the seventh grade students for the most part could use the Simulation Creation Toolkit and follow the steps necessary to create a given Predator/Prey simulation in 4 days. It should be noted that not having correct patterns does not necessarily mean that the simulation is completely incorrect depending on what is incorrect. For example, if a Rabbit Agent is allowed to move onto other Rabbit Agents, that agent will still get hungry, pursue food, eat die, and mate. In fact, the only problem is that if another Rabbit Agent is on top of it, the Dirt Agent will not be able to access it.

5.1.3 Simulation Totals For The Sixth Grade Computer Class

In contrast to the seventh grade students, the sixth grade students had little prior programming experience, simulation experience, and received this simulation out of context as, unlike the seventh grade students, they were not studying ecosystems at the time. Furthermore, instead of 4 days to complete the simulation, the sixth grade students had 3 days. The sixth grade class was included to see how students who had never used AgentSheets/AgentCubes responded to the Simulation Creation Toolkit. Both teachers estimated, as the worksheets were being split up into days, that 3 days would place students at Pattern 10 wherein students implement the

Rabbit Agent tracking the Grass Agent (see Pattern 10 in section 4.2.1). Therefore, in addition to the full analysis of the sixth grade students submissions an additional analysis of consisting of Patterns 1-10 will also be used. This analysis will better allow comparisons with the seventh grade class. Data was taken from 21 sixth grade students over 2 class periods.

The following radar chart depicts the partial credit pattern analysis for all sixth grade students.

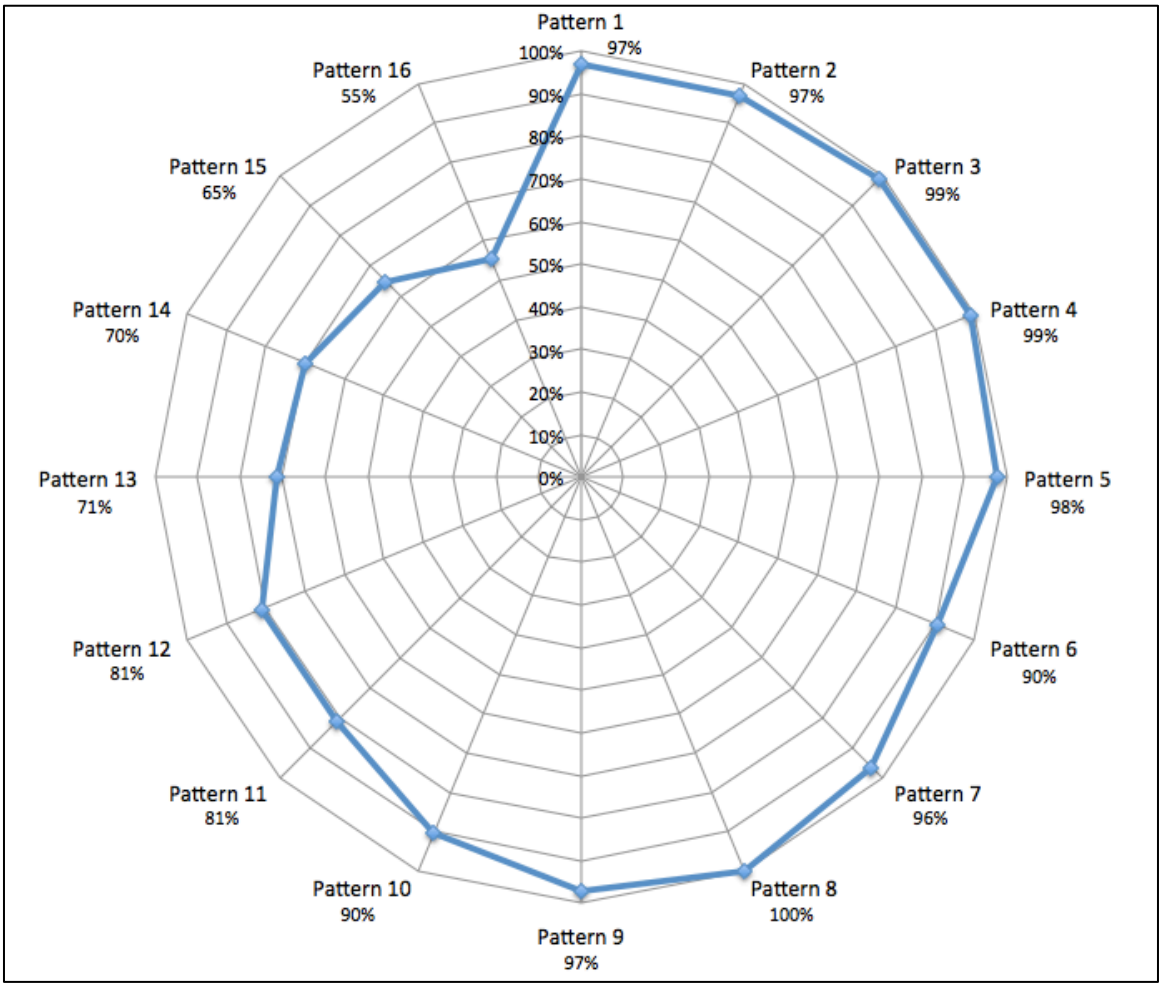


Figure 88: Final Partial Credit Pattern Averages For All Sixth Grade Students

As to be expected, the percentages in Figure 88 start dropping off around Patterns 9 and 10. Also to be expected, the lowest percentage is 55% at Pattern 16. The following figure depicts the partial credit pattern averages for all sixth grade students excluding the patterns after Pattern 10.

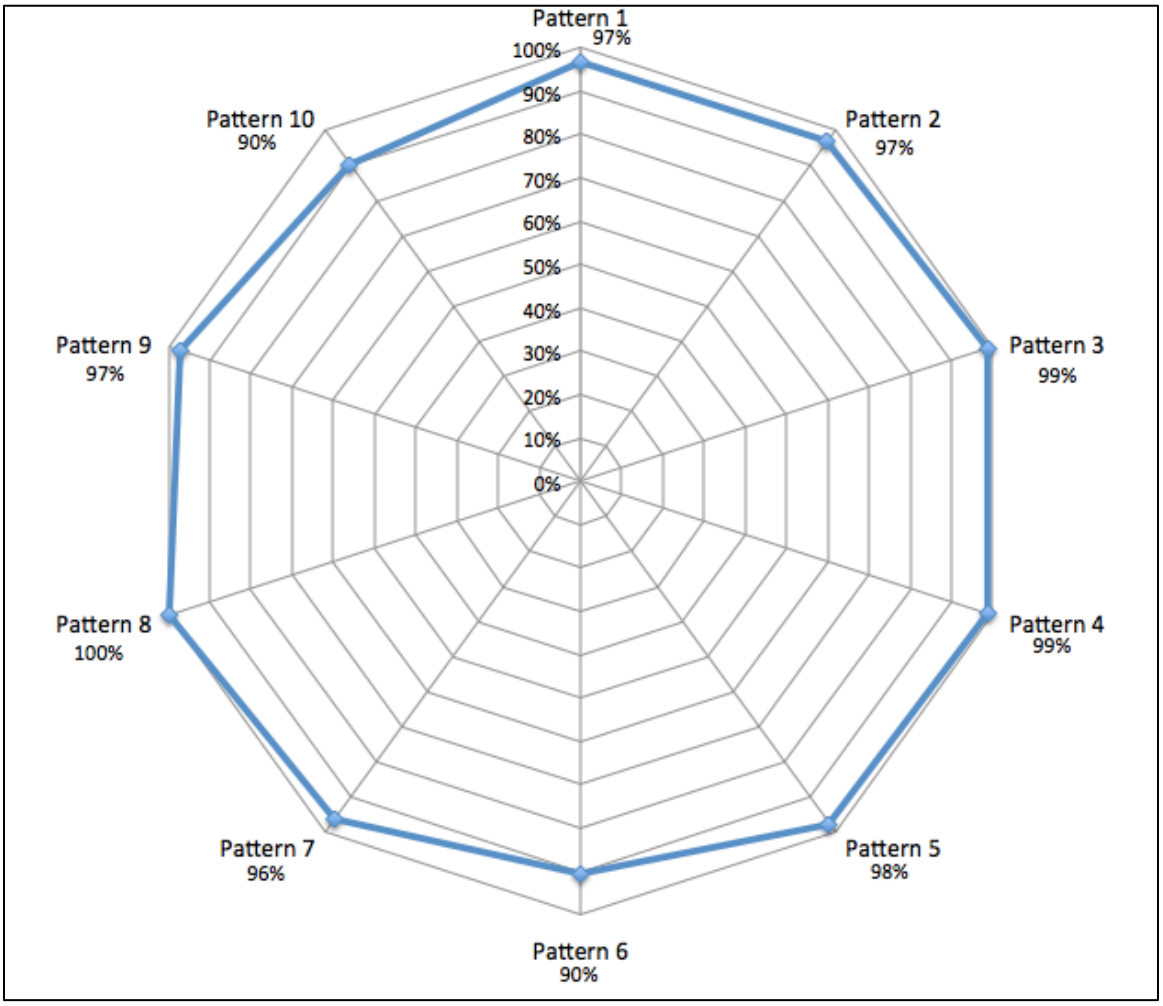


Figure 89: Pattern 1-10 Partial Credit Pattern Averages For All Sixth Grade Students

From Patterns 1-10 the sixth grade students did extremely well, with the least percentage being 90% at Pattern 10. This could be due to the fact that some students did not reach Pattern 10 with enough time to implement it by the end of day 3. To further display how sixth grade students compared to

seventh grade students, the following radar chart compares both the sixth and seventh grade students' simulations from Patterns 1-10.

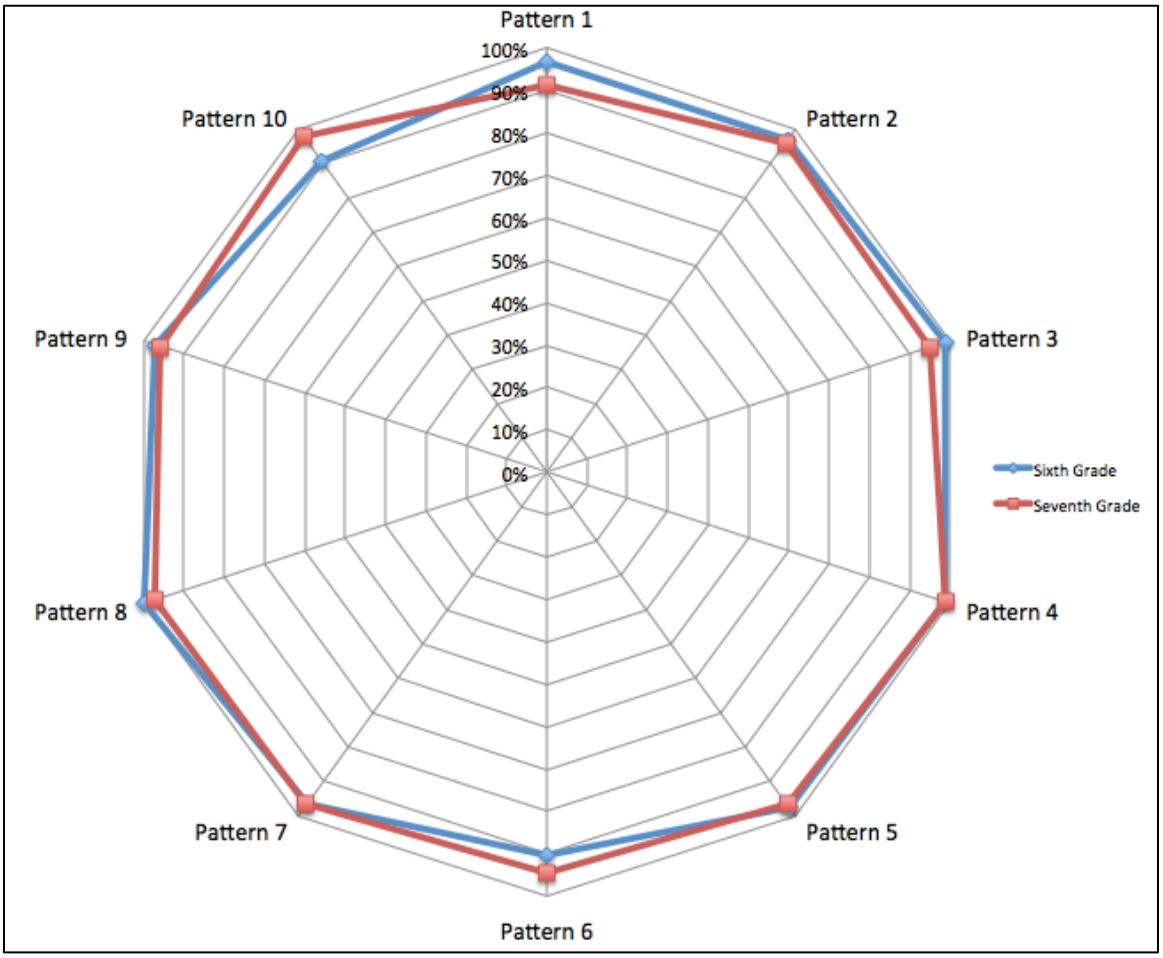


Figure 90: Comparison Of Sixth And Seventh Grade Student Simulation Partial Credit Pattern Averages For Patterns 1-10

Figure 90 shows that there is very little difference between students who had never used AgentSheets/AgentCubes before and students who had prior experience. In fact, for Pattern 8, Pattern 3, and Pattern 1 the sixth grade students actually did better according to the partial scores than the seventh grade students. There are a few reasons this might be the case. First of all, the sixth grade students had the lesson second; therefore, the teaching strategy was much improved over the first week of lessons. Concepts that

might have been confusing to students the first week such as pattern specifications were better explained the second week to the sixth graders early on. Specifically, many students in the seventh grade class missed the blocking agent specification for the Random Movement Pattern in Pattern 1. This idea could be better emphasized in the second week to the sixth grade students. Furthermore, the population of sixth grade students included in the study might not have been completely representative of the whole sixth grade class. For example, Marco Cornacine had almost a month to get students to return their permission forms yielding many more students returning their permission forms (though many of the students who returned their forms had data taken out because they did not turn in worksheets or missed a day of simulation building). Since Marks Savs's class works on the quarter system, and since the quarter had just began, students had one week to return their forms. This could have led to the sixth grade class being self selected students. It is impossible to know to what extent this is true; what is known is that about half of Marco's students had their data analyzed and about one third of Mark's students had their data analyzed.

Looking at this data it is clear that sixth grade students with no programming experience or simulation experience can indeed use the Simulation Creation Toolkit to execute the steps necessary to create correct pattern implementations which is essential for creating working simulations using the tool. The following graph depicts the all or nothing credit for sixth grade students over all 16 patterns.

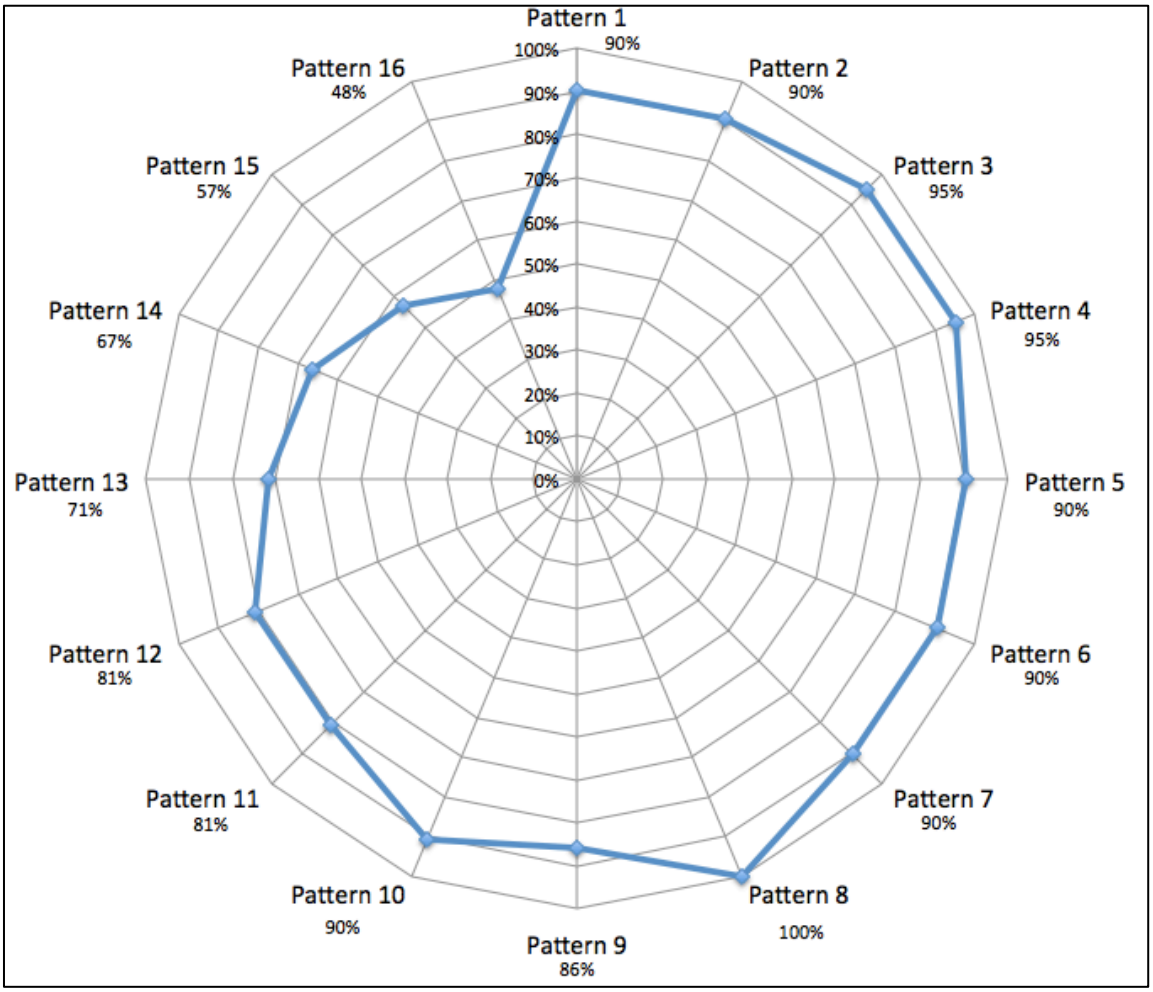


Figure 91: Final All Or Nothing Credit Pattern Averages For All Sixth Grade Students

As with the Partial Credit Pattern Averages the percentages for the all or nothing credit pattern averages start to decline after Pattern 10. However, almost 50% of students were able to correctly implement Pattern 16. The following graph is the all or nothing credit of the sixth grade class as compared to the seventh grade class for the first 10 patterns.

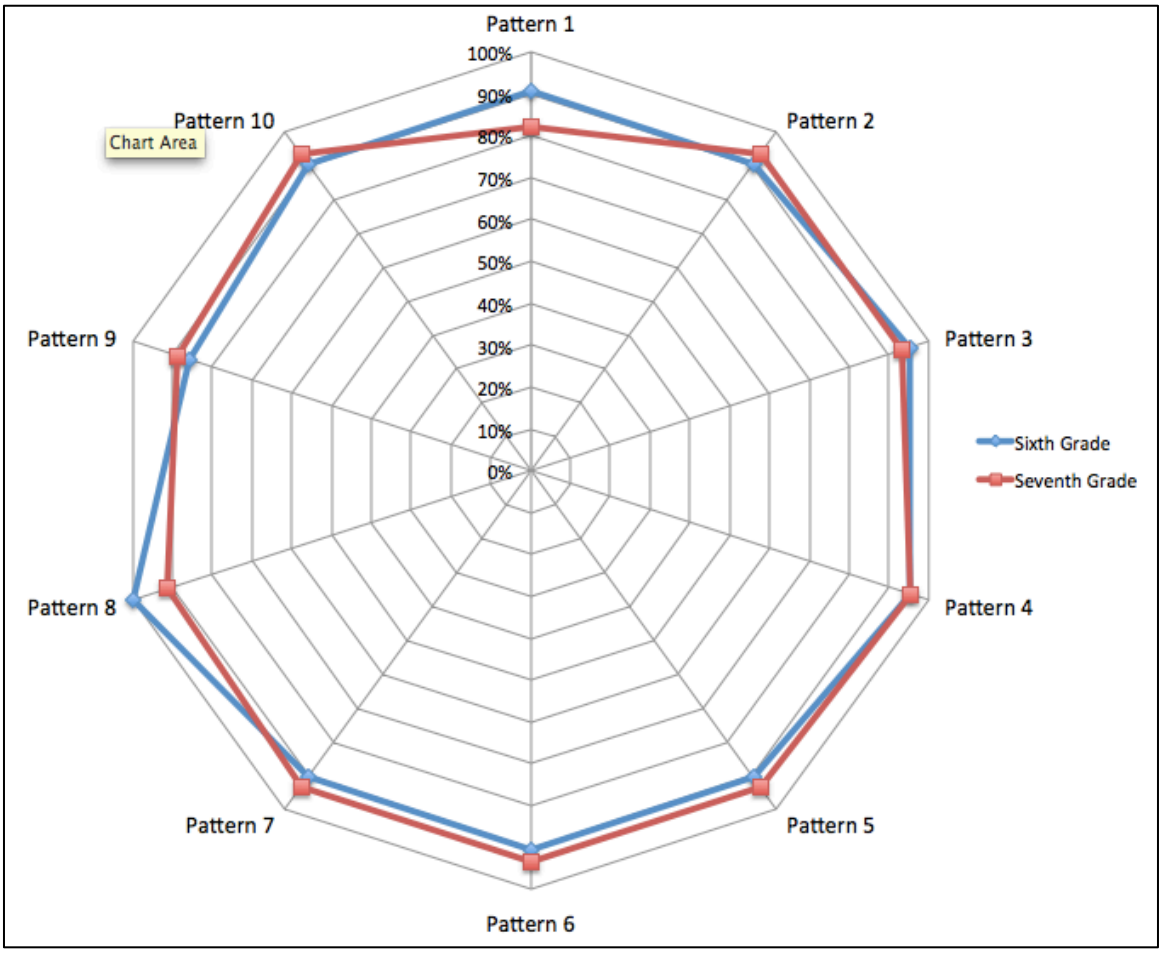


Figure 92: Comparison Of Sixth And Seventh Grade Student Simulation All Or Nothing Credit Pattern Averages For Patterns 1-10

As with Figure 90, the all or nothing credit pattern averages depicted in Figure 92 shows that the sixth grade students were comparable to the seventh grade students through the first 10 patterns. For the most part, the percentages are almost identical. This implies that given an extra day, the sixth grade students could have very similar results to the seventh grade students for all the patterns necessary to create the Predator/Prey simulation. This data indicates that the sixth grade student shortcomings, which included little experience with the concepts of the unit itself or simulation creation in general, could be overcome in a highly scaffolded environment using the Simulation Creation Toolkit.

The following table categorizes sixth grade students into how many patterns they implemented correctly.

	Perfect Simulation	1 Incorrect Pattern	2 Incorrect Patterns	3 or More Incorrect Patterns
Number Of Students	5	2	2	12
% Of Students	23.8%	9.5%	9.5%	57%

Table 10: Categorizing Sixth Grade Students By How Correctly They Implemented The Predator/Prey Simulation

According to Table 9 almost a quarter of sixth grade students were able to create the Predator/Prey Simulation correctly. Almost, 43% of students had 2 or less incorrect patterns. However, 57% of sixth grade students also had 3 or more incorrect patterns. This probably has a lot to do with the limited amount of time the sixth graders had to create the simulation; as many of them did not do the latter patterns. Still, in order to have only 2 or more incorrect patterns implies that students had to at least get to Pattern 14. The following table categorizes sixth grade students into how many of the first 10 patterns (Patterns 1-10) they implemented correctly.

	Perfect Patterns 1-10	1 Incorrect Pattern	2 Incorrect Patterns	3 or More Incorrect Patterns
Number Of Students	10	7	1	1
% Of Students	47.6%	33.3%	4.8%	4.8%

Table 11: Categorizing Sixth Grade Students By How Correctly They Implemented Patterns 1-10 In The Predator/Prey Simulation

These results seem closer to the seventh grade class percentages. Almost half of the sixth graders had a perfect simulation by the end of Day 3.

Furthermore, about 85% of sixth grade students had only 2 incorrectly implemented patterns after implementing the first 10 patterns. Therefore 85% of sixth grade students were on pace to finish the simulation by the end of Day 3 with about half the students on pace to complete it perfectly.

5.2 Worksheet Response Results

As with the Simulations collected, results from 66 student worksheets were taken and analyzed. This study used a scoring system similar to the one presented in *Classroom Assessment For Student Learning* by Chappius et al. wherein each question is scored on a scale from 0-3 [54]. The following table describes generally how the score relates to the question answer.

Points	Level of Understanding
3	Shows complete understanding
2	Shows partial understanding with few, if any, simple misunderstandings
1	Shows partial understanding with some misunderstandings but no "fatal flaw"
0	Evidence shows lack of understanding, misunderstanding, or partial understanding with inclusion of a "fatal flaw"

Table 12: How The Questionnaire Answers Relate To The Scores [54]

Questions that were not attempted by students are not included in the analysis. The first part of this analysis will review each question and show how the sixth and seventh graders each did on each question. The second part of this analysis will combine questions into categories to better

understand how students performed on a certain group of questions. For a summary of the questions and what might be considered right answers see section 4.3.2. Kyuhan Koh, Ph.D. student in Computer Science at the University of Colorado Boulder, helped calibrate the questionnaire correction.

5.2.1 Question By Question Worksheet Results

What follows is a question by question breakdown of the worksheet results.

Question 1: Question 1 simply asks students to open the simulation properties in AgentCubes and record the initial populations of the Fox Agent, the Rabbit Agent, and the Grass Agent. This question is pretty simple and every student correctly answered it.

Question 2: Question 2 asks students to open up the Simulation Creation Toolkit and asks what the three data patterns that have already been implemented. The hope is that students make the connection between the simulation properties displaying the agent populations and the three patterns already implemented. In a big picture sense, it is an opportunity for students to make the connection between the idea that the patterns they implement using the Simulation Creation Toolkit have a direct effect on the simulation they are programming. As mentioned in 4.3.2, this question might be a little ambitious given that students have just learned about patterns and the Simulation Creation Toolkit just a few minutes prior. The following graph depicts the score breakdown of 7th grade students' answers to Question 2.

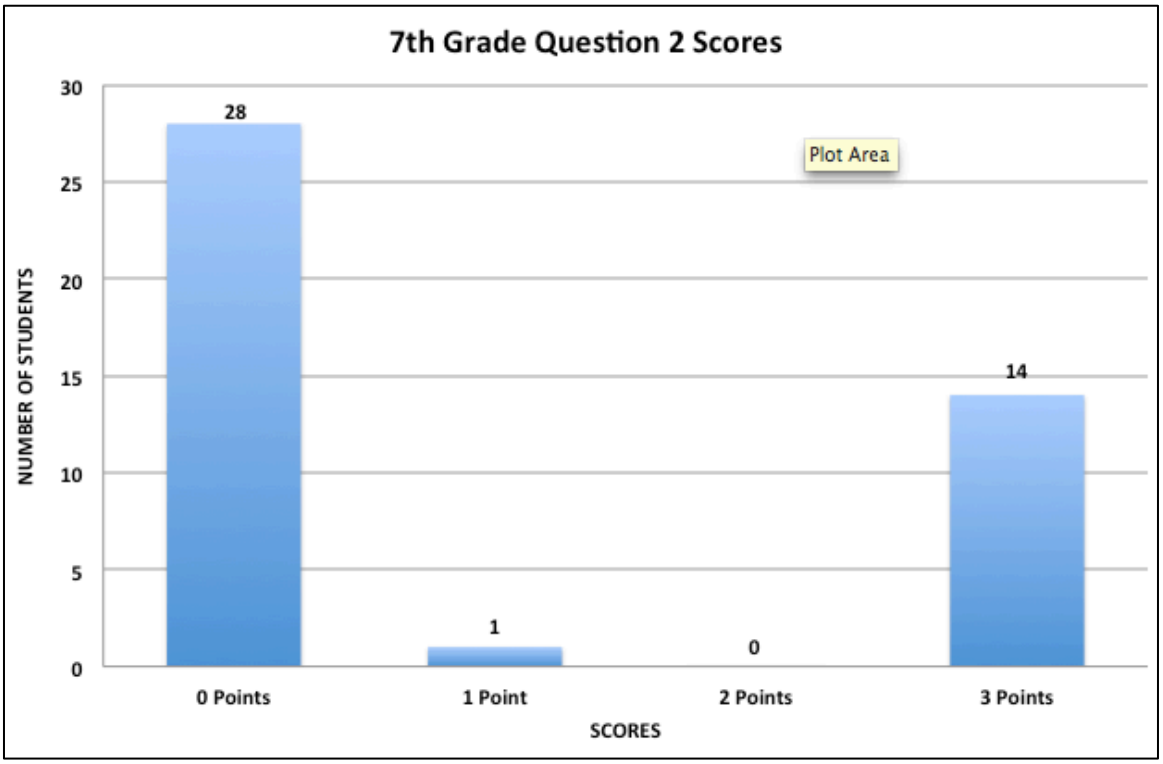


Figure 93: 7th Grade Students Question 2 Scores

As one can see from the above graph, a little less than a third of the 7th grade students answered this question correctly and almost two-thirds of students answered this question completely wrong.

The following graph depicts score breakdown of the 6th grade students' answers to Question 2.

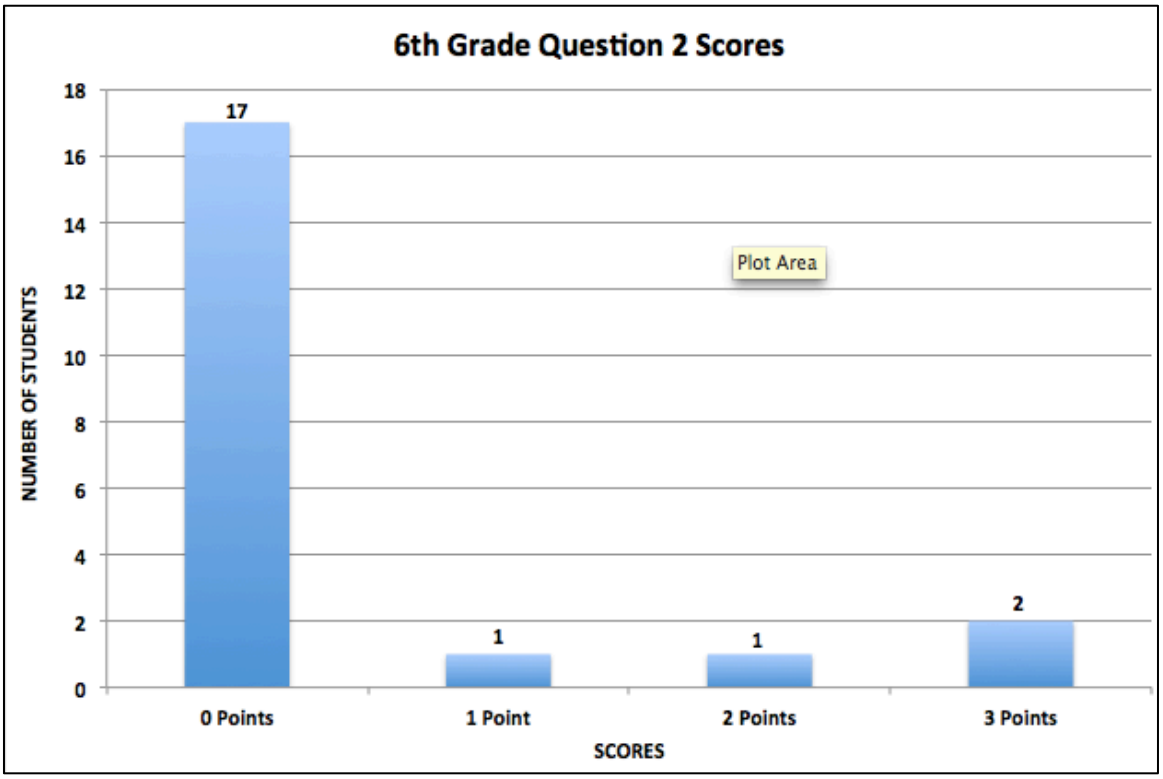


Figure 94: 6th Grade Question 2 Scores

Only 4 sixth graders were able to make any connection between the already implemented patterns and the Simulation Properties counting the agents. From both the 6th and 7th graders one can surmise that a better introduction to the tool might be necessary to have students make this connection immediately as it is not readily apparent.

A response that was scored a 3 was able to associate the already implemented patterns with the functionality of the simulation to count these agents when run. A typical 3 point response is as follows:

"Count the number of certain agents on the screen."

Most students however, were confused by the question. The following quotes were typical student answers to this question that were scored a one or a zero.

"They put things on the screen."

"They help make the simulation accurate."

"I think the patterns control what the agents are doing and how they are doing it"

"I don't know. ☹"

The idea that students would be able understand how the patterns relate to the simulation before they actually implemented a pattern themselves might have been an unrealistic expectation. Also, as mentioned above, the introduction could have drawn this concept more clearly. Finally, it might be necessary for future iterations of the Simulation Creation Toolkit to some sort of mechanism that shows what rules the rules do and/or how they relate to the implemented patterns more clearly.

Question 3: Question 3 asks students to give a reason why the random movement might be unrealistic. This question is one of the first questions that ask students to think critically about the simulation they are creating. Namely, this question sets up the idea of representational systems not necessarily representing the real world domain exactly, but representing it to the point wherein one can explore the problem domain more effectively (see Figure 1). The correct answer to this question would make reference to the idea that these animals would not actually move randomly around all the time.

The following graph depicts the breakdown of 7th grade students' answers to Question 3.

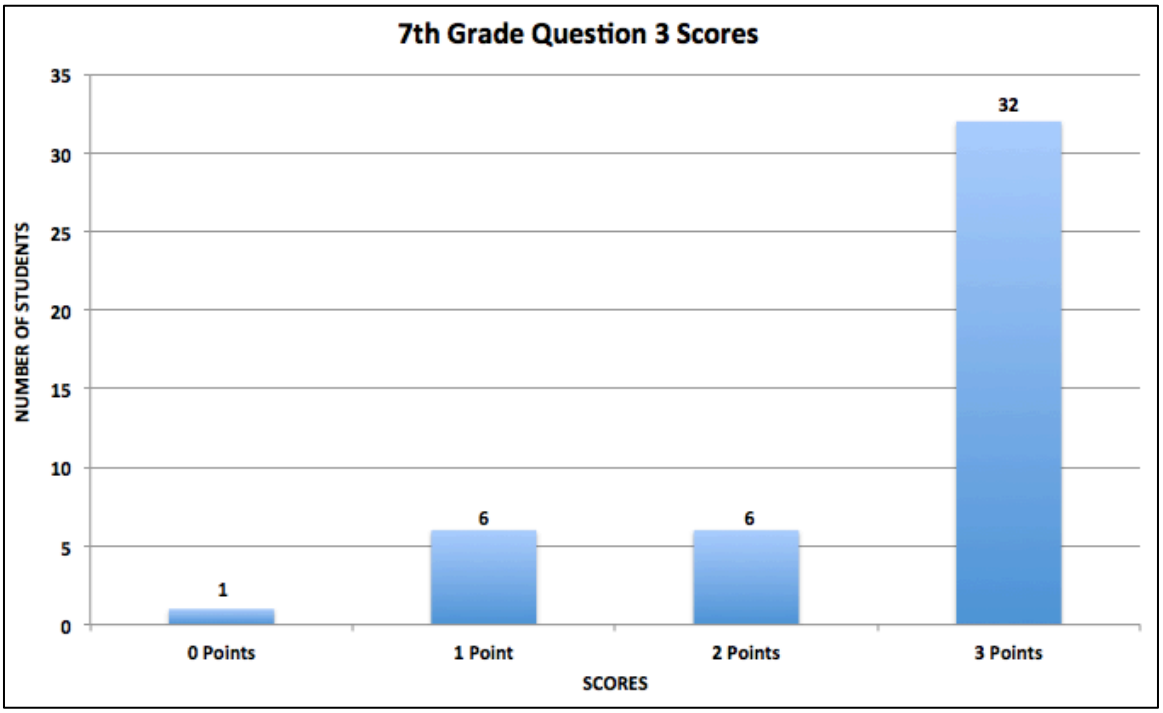


Figure 95: 7th Grade Question 3 Scores

Figure 95 shows that most 7th grade students answered this question correctly. Unlike the previous question, this question is easy to understand and once students understand it, it is not hard to produce a single reason that this movement might be unrealistic. Furthermore, 7th grade students had previously been exposed to using simulations a few times earlier in the year (see section 1.3.2) , and thus had explored the idea of models before and may have been more perceptive to assumptions of a model that could be unrealistic.

The following graph depicts the breakdown of 6th grade students' answers to Question 3.

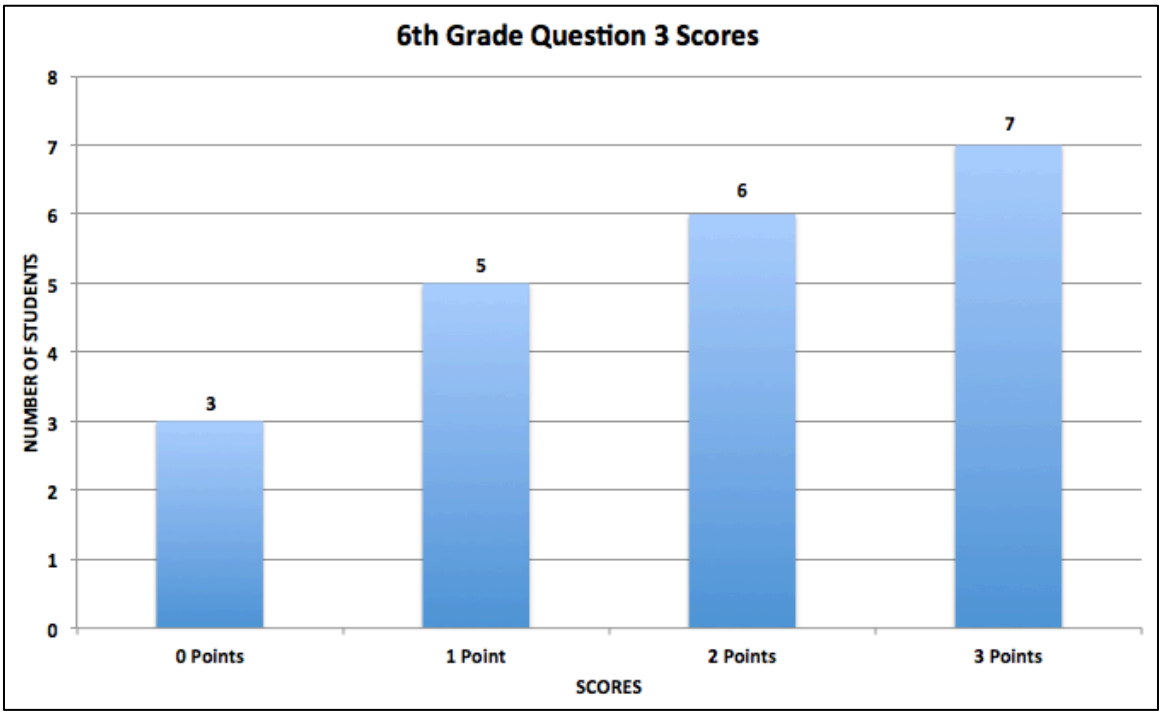


Figure 96: 6th Grade Question 3 Scores

Unlike the 7th grade students, the 6th grade students had an higher percentage of partially incorrect responses. This might be due to the fact that the use of simulations was new to these students and thus some students might not have immediately fully understood this concept.

The following answers were typical correct answers for this question.

"They wouldn't be moving randomly they would probably be moving in a certain direction…"

"The Fox and Rabbit won't move randomly, but they will have a destination…"

Students who got this question right seemed to understand that the animals would probably move with some sort of purpose rather than randomly. The following answers were typical of partially correct answers.

"Animals don't roll dice in real life."

"The fox doesn't eat the rabbit."

The first answer refers to the introduction exercise that took place in the class wherein students rolled dice to see which direction they moved in (see section 4.2.1). This answer is on the right track in that animals do not move randomly, however the wording is such that it is not clear whether the student understands this idea or rather is being explicit in their interpretation of animals rolling dice. The second answer, referring to the Fox Agent not eating the Rabbit Agent is true but is not a reason why the random movement is unrealistic; for example, the Fox Agent could move randomly and eat the Rabbit Agent and its movement would still be unrealistic.

Question 4: This question asks students what pattern they would use to have the dirt transform a Fox Agent into a Hungry Fox Agent. Since this creation exercise must be heavily scaffolded in the classroom environment, a few questions ask students to look at the Simulation Creation Toolkit palette of patterns and decide what pattern they would use. There are a few correct ways to answer this question. Some students said "collision" which is technically correct because the general collision pattern category, as defined in the Simulation Creation Toolkit, involves something happening when two agents come together. Other students said the "Change Pattern" which is correct, as the Change Pattern is the specific collision pattern that would be used to change a Fox Agent into a Hungry Fox Agent. Finally, some students described the pattern that they would use which is also a correct way to answer the question as it indicates they know the type of interaction in the Simulation Creation Toolkit they would look to implement.

The following graph depicts the breakdown of 7th grade students' answers to Question 4.

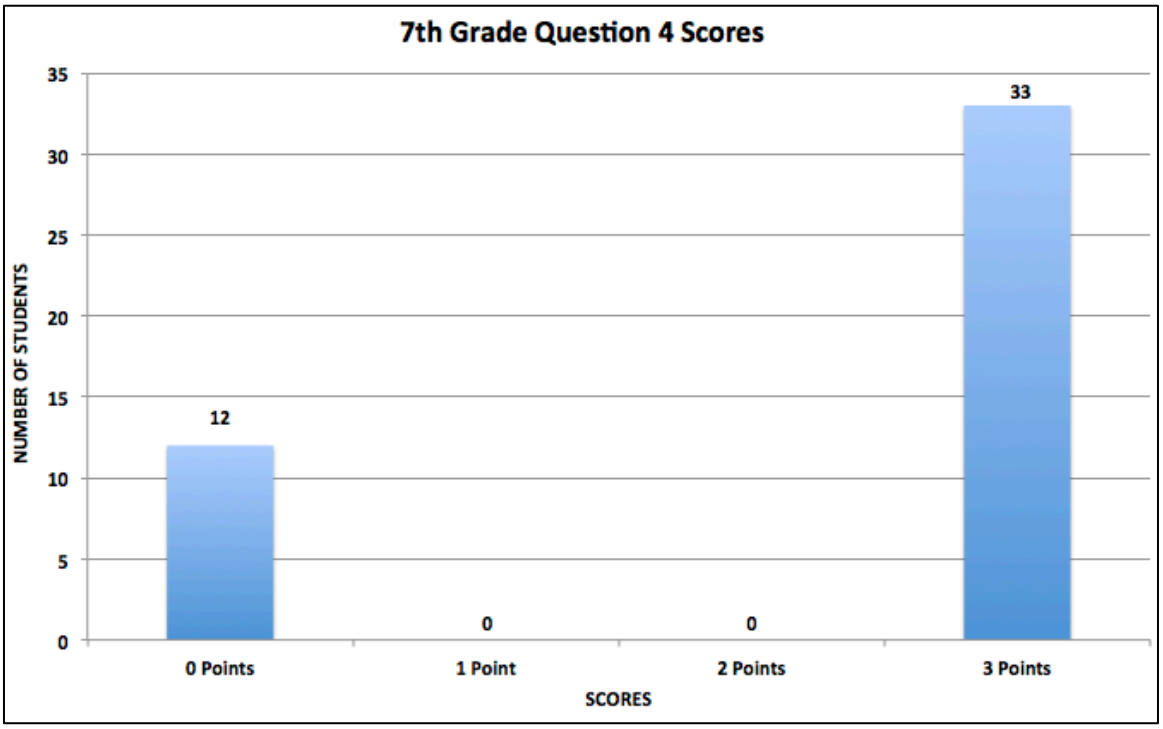


Figure 97: 7th Grade Question 4 Scores

Most seventh grade students were able to name or describe the correct pattern to use from the Simulation Creation Toolkit. Almost a quarter of the students had no idea the correct pattern to use.

The following graph depicts the breakdown of 6th grade students' answers to Question 4.

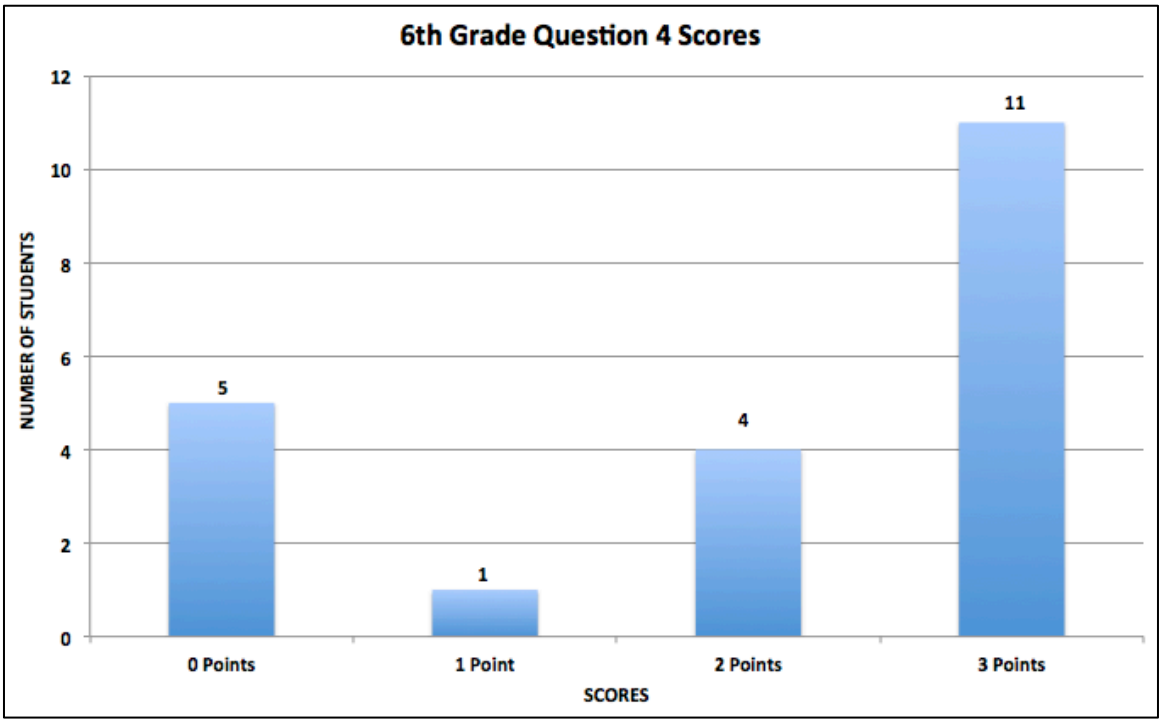


Figure 98: 6th Grade Question 4 Scores

About half of the 6th grade students were able to reason out the correct pattern to use for this interaction. It is hard to say exactly why the 6th grade students had more trouble with figuring out which pattern to use. One reason might be that the 6th grade students had never been exposed to agent interactions before. For example, 7th graders had made Frogger in AgentSheets the previous year, therefore, the idea that the Truck agent would collide with the Frog Agent, for example, changing the Frog Agent to a Dead Frog would not be new. This interaction between the Dirt Agent and the Fox Agent is in many respects similar to that interaction. For the 6th grade students, however, the idea of agents colliding and one agent being changed into another agent was a brand new concept. Therefore, these students had to fully rely on the ability to navigate the Simulation Creation Toolkit to find a pattern they might use for this interaction.

The following are typical correct answers for Question 4.

"The collision pattern, because something will happen when they collide"

"change one agent into another."

"I think we will use collisions-change."

These answers all describe the correct general pattern or specific pattern (or both) that will enable the Dirt Agent to change the Fox Agent into a Hungry Fox Agent.

The typical wrong answers usually named the wrong pattern. The following are typical of incorrect answers for this question.

"I think we might use the Generation Pattern."

"Tracking so the Hungry Fox can eat the Rabbit."

For the first answer, it could be that the student does not completely understand that the Fox Agent is being changed into a Hungry Fox Agent and not that a Hungry Fox Agent is being generated. The second response is correct in terms of the simulation but has nothing to do with the Dirt Agent changing the Fox Agent into a Hungry Fox Agent.

Question 5: Question 5 attempts to have students understand that energy is being used up as the animals move around. This is meant to reinforce the idea that energy exists in these ecosystems, which is a GVC concept that students often overlook, and motivate the idea of agents becoming hungry in the simulation. The correct answer would make reference to the term "energy."

The following graph depicts the breakdown of 7th grade students' answers to Question 5.

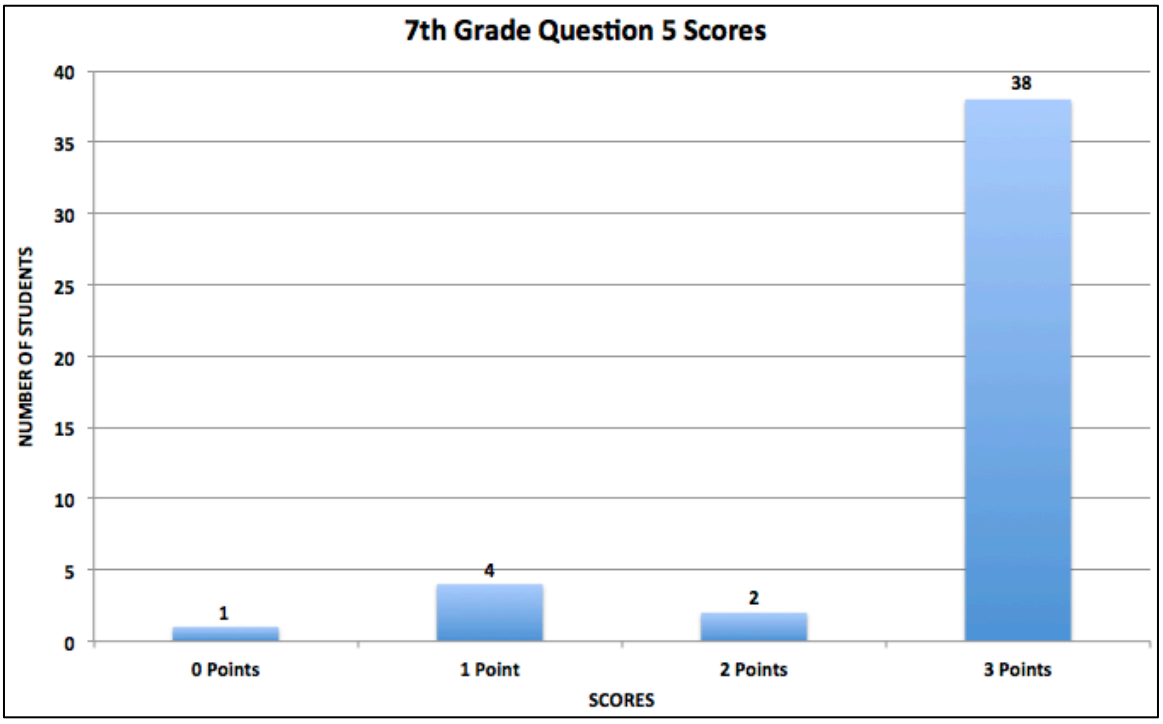


Figure 99: 7th Grade Question 5 Scores

Almost all 7th grade students understood the idea that energy being expended as animals move around. The 7th grade students had a semester wherein the ideas of energy consumption were part of the class curriculum. Interestingly though, most 6th grade students got this question correctly too.

The following graph depicts the breakdown of 6th grade students' answers to Question 5.

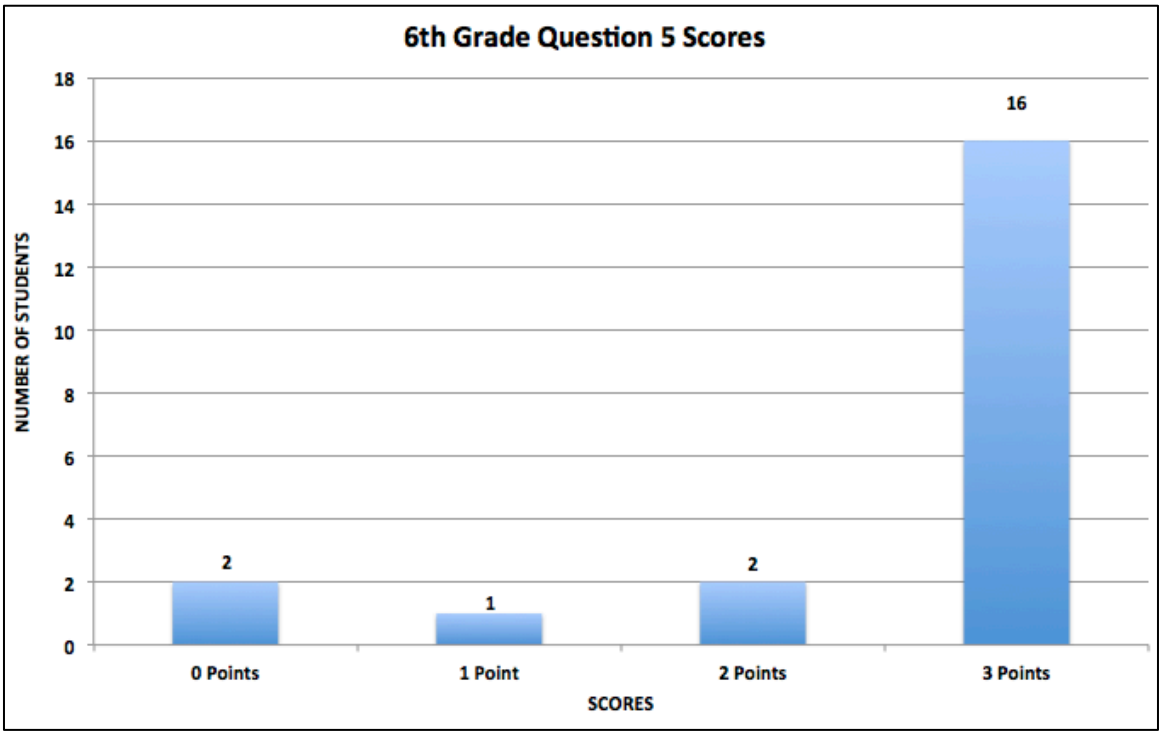


Figure 100: 6th Grade Question 5 Scores

Given that this idea was something that Marco Cornacine thought should be covered in the simulation for the 7th grade students, it interesting to see that most 6th grade students already understood this idea. The simulation itself does not explicitly refer to energy in any way other than having agents turn into hungry agents. It could be that this concept enabled students, regardless if they had learned this concept over the year in their respective classes or not, to make the connection between energy expenditure and movement. It could also be possible that the 6th grade students had also been exposed to this concept in their other classes.

The following quotes are typical correct responses to this question.

"They are using up energy which makes them hungry."

"They are using energy up as they move."

The first answer alludes to the above idea wherein students might see that the animal gets hungry and therefore infer that the animal is using up

energy as it moves. Incorrect and partially correct answers often tried to draw from the simulation itself as to what was being used up. The following are typical partially correct and incorrect answers to Question 5.

"They are using up the rest of the percentage until it says they have changed."

"As the fox and rabbit move, they are using up time."

"Their chance to not be hungry."

The first response and the last response are referring to the idea that as these agents move, there is a percent chance that they will become hungry. The middle response is the idea that time elapsing as these agents move. In some sense, these answers are not entirely incorrect, they simply do not connect a pattern in the simulation world with what it is representing in the real world.

Question 6: Asks students if a given Fox and Rabbit Agent are more or less likely to become hungry as the simulation runs. The idea behind this question is that the Dirt Agent is changing them into Hungry Agents with a percent chance every so often. Therefore, if an agent is not hungry, as the simulation continues they are more likely to become hungry. Furthermore, as the simulation the Agent is moving and thus expending more energy as time goes on. Thus, the simulation is realistic in the sense that Agents are moving and as they move they will eventually get hungry in the simulation. Any answer along these lines is deemed correct. This question is an example of the representational system implementing something in the real world domain realistically.

The following graph depicts the breakdown of 7th grade students' answers to Question 6.

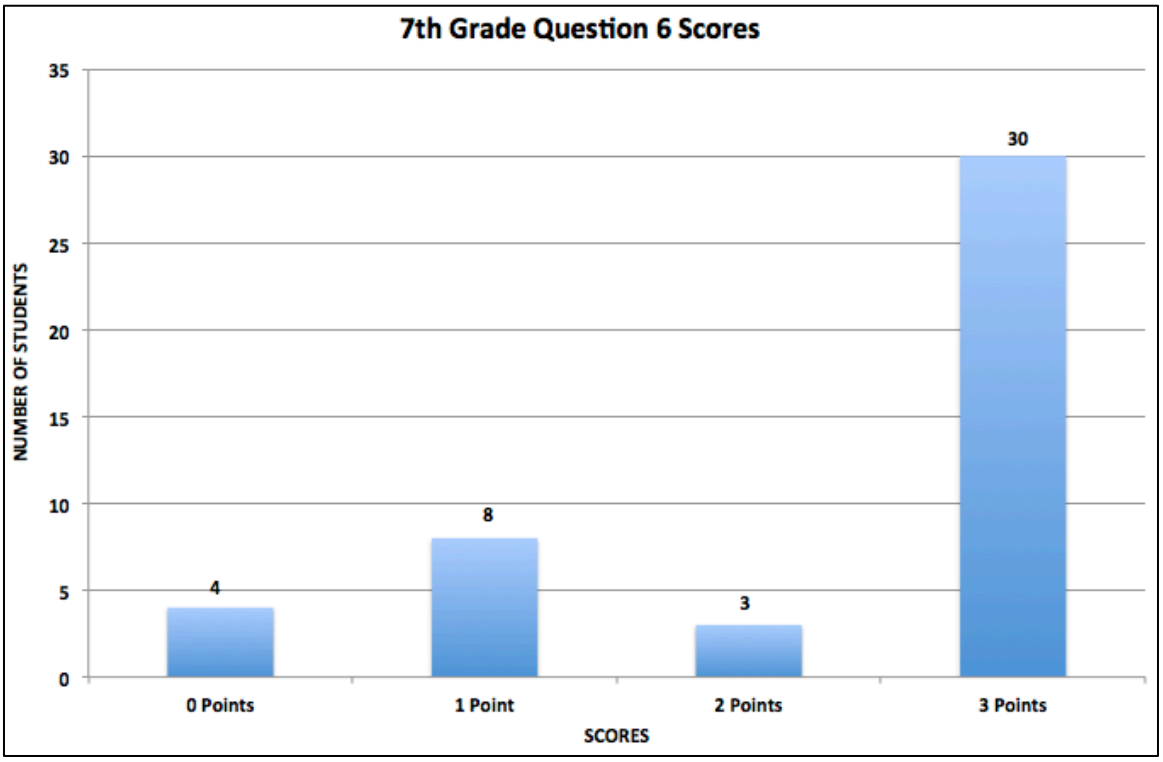


Figure 101: 7th Grade Question 6 Scores

Most 7th grade students got this question correct. This question necessitates the need for students to connect the mechanism the Simulation Creation Toolkit uses to implement a given interaction, though not realistic in and of itself, yields correct behavior in the representational system of the real world. Specifically, dirt in real life would not change a fox into a hungry fox. However, students seem to understand that having the Dirt Agent in the simulation change the Fox Agent into a Hungry Fox Agent with a given percent chance leads to the Fox Agent eventually getting hungry over time and thus is a somewhat realistic representation. This is necessary as the Simulation Creation Toolkit relies on students using a limited amount of patterns, therefore, to achieve certain behaviors necessitates the ability to use patterns creatively to get a given interaction between agents. Similar to

the 7th grade students, most 6th grade students were able to reason out the correct answer to this question.

The following graph depicts the breakdown of 6th grade students' answers to Question 6.

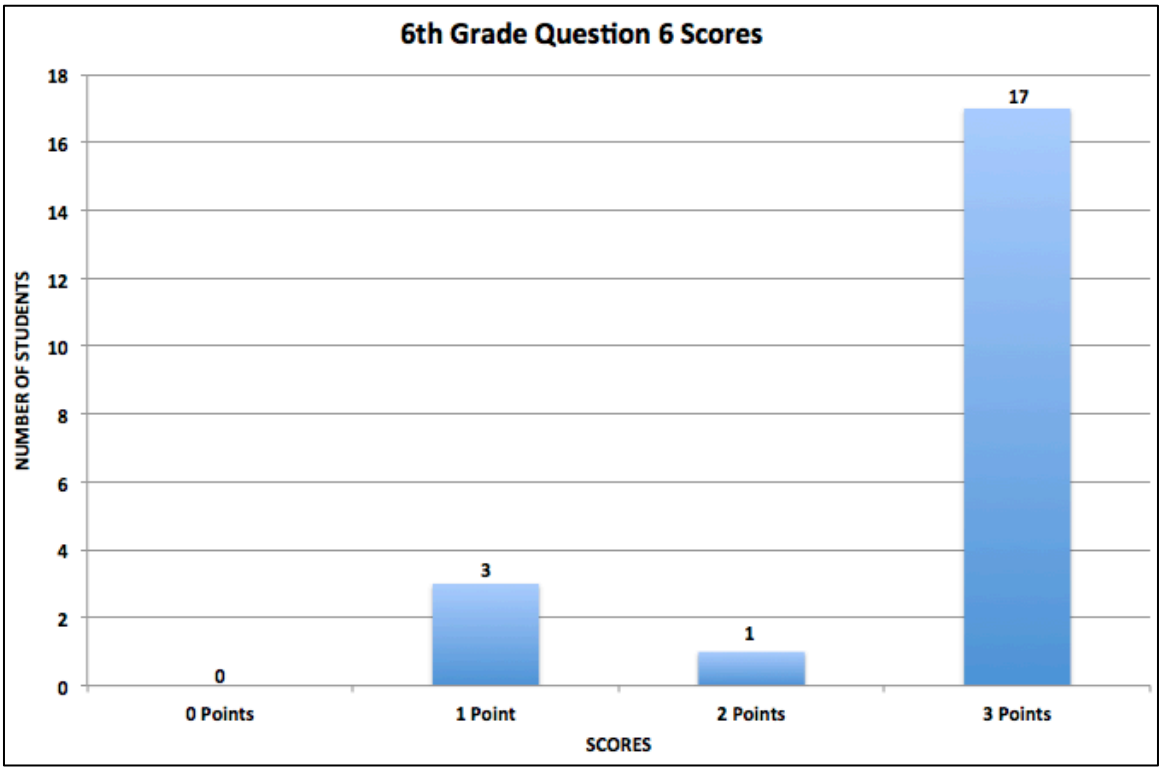


Figure 102: 6th Grade Question 6 Scores

The following are some typical correct responses to Question 6.

"More likely to become hungry. It is realistic because the more they move the hungrier they get."

"They are more likely to get hungry because they are using energy. Yes it is realistic."

Both of these answers show that students understand that in the simulation they have created so far, the agents are more likely to get hungry as time goes on and that this is realistic as they are also moving (or using up energy)

as time goes on. The wrong answers tended towards "I do not know" or an answer to a different question (i.e. the student may have misread the question). Therefore were not that informative.

Question 7: Question 7 asks students what the Hungry Fox and Hungry Rabbit Agents should do instead of stop moving. The correct answer is find their food (Rabbit Agent for Fox Agents and Grass Agent for Rabbit Agent) or move towards food. Many students went a step further and used the word "Track" or "Tracking Pattern" taken from the palette of patterns in the Simulation Creation Toolkit. This question was meant to motivate the next steps in the simulation creation exercise; the fact that many students used the specific name of the pattern is exciting as it means students were thinking in terms of how to implement the interaction using the Simulation Creation Toolkit palette.

The following graph depicts the breakdown of 7th grade students' answers to Question 7.

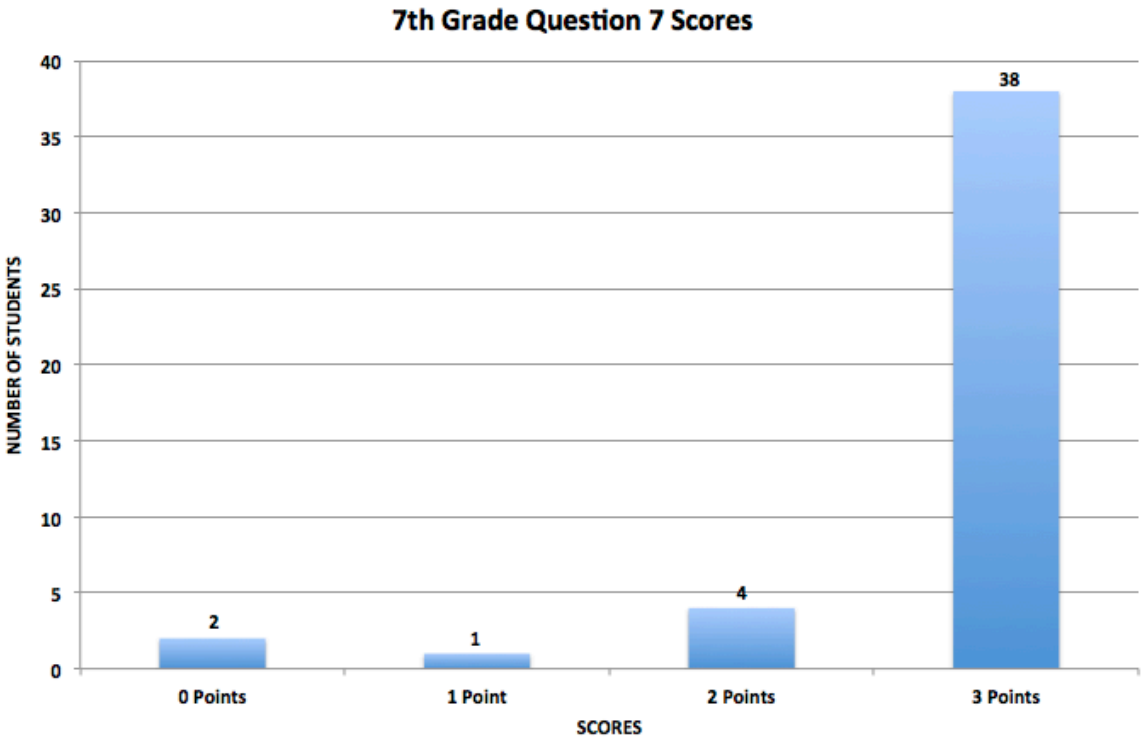


Figure 103: 7th Grade Question 7 Scores

Most 7th grade students were able to get this question correct. This is probably in part due to the fact that the question is somewhat obvious; namely, hungry agents should find and eat their respective food. Similarly, most 6th grade students got this question right too.

The following graph depicts the breakdown of 6th grade students' answers to Question 7.

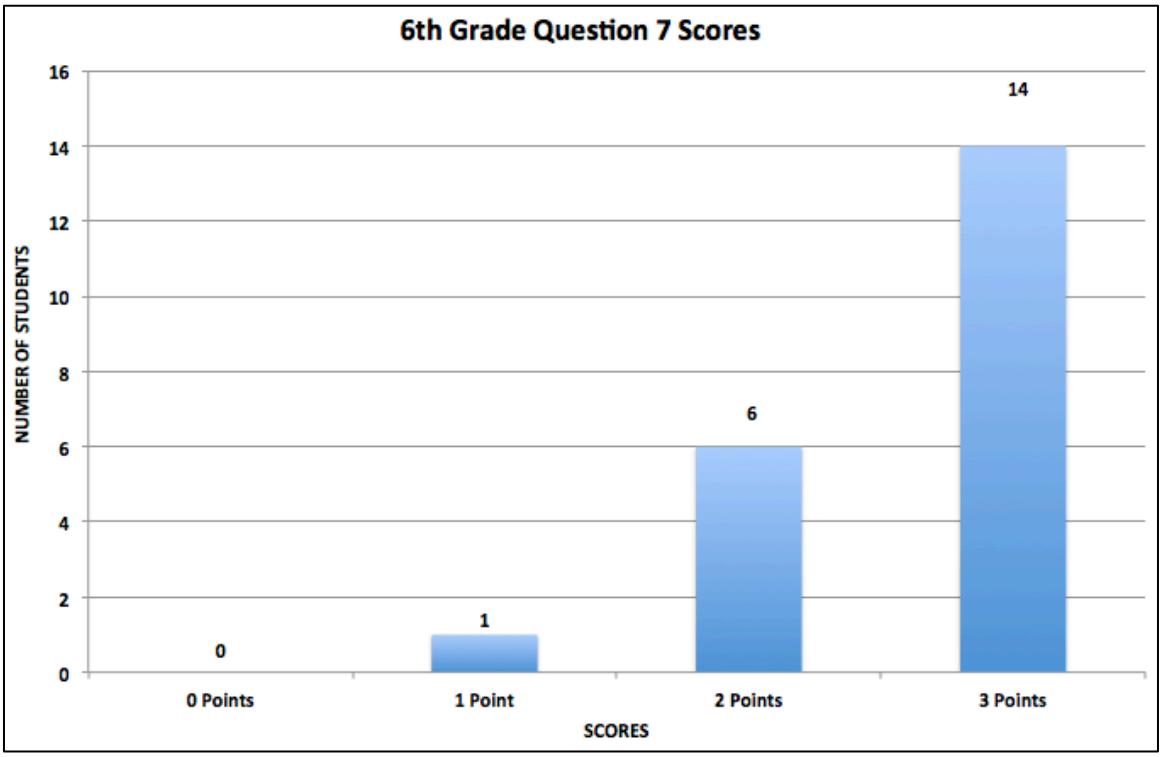


Figure 104: 6th Grade Question 7 Scores

Again, the idea of a Hungry Agent needing to track food seemed obvious to both sixth and seventh grade students.

The following quotes are representative of typical correct responses to Question 7.

"The hungry fox would track the rabbit, and the rabbit would track the grass"

"The hungry fox dies or goes towards the rabbit, same with the rabbit but towards the grass."

"They both should start looking for food/tracking."

The first answer and the third answer mention tracking specifically, and as mentioned above, this was pretty typical for an answer to this question. The middle answer goes above and beyond in the way that the student realizes the Hungry Agent should die if it does not find food or go after their respective food.

The partial credit answers and incorrect answers either focused on the agents dying or referenced a pattern and interaction that did not fit into the context of the question. The following two quotes are typical of these responses.

"They should turn into a dead fox or rabbit."

"They need to mate"

The top response, though partially correct does not fully explain what the Hungry Agent should do in the simulation. The second answer is wrong in terms of this question, but it is true that these agents eventually need to mate. However, mating is not really the specific interaction we are looking for when asking a question about the Hungry Agent.

Question 8: Question 8 refers to the idea of how quickly the Fox Agents and Rabbit Agents decompose in the simulation as compared to how quickly they decompose in real life. Correct answers referenced the idea that the Fox and the Rabbit Agent decompose into grass much more quickly in the simulation than they do in real life. The general idea behind this is that decomposition takes a long time.

The following graph depicts the breakdown of 7th grade students' answers to Question 8.

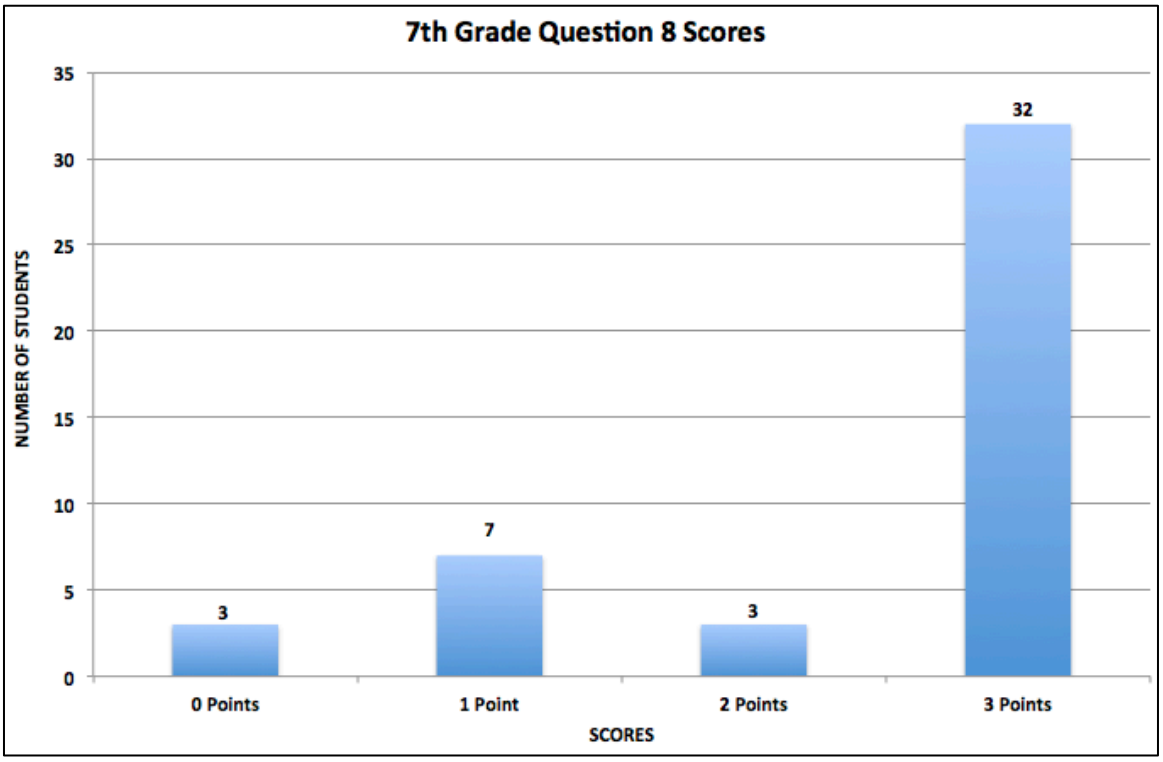


Figure 105: 7th Grade Question 8 Scores

Most 7th grade students were able to get this question correct. Given the way the question was phrased students knew that the answer had to do with the time of decomposition. From this it seems it was easy for students to reason out that the decomposition time in the simulation was unrealistic based on the time it took for an agent to decompose. Furthermore, given that the 7th graders had learned that decomposition takes a long time, many could deduce the correct answer. Amazingly enough, 6th graders did even better on this question than the seventh graders.

The following graph depicts the breakdown of 6th grade students' answers to Question 8.

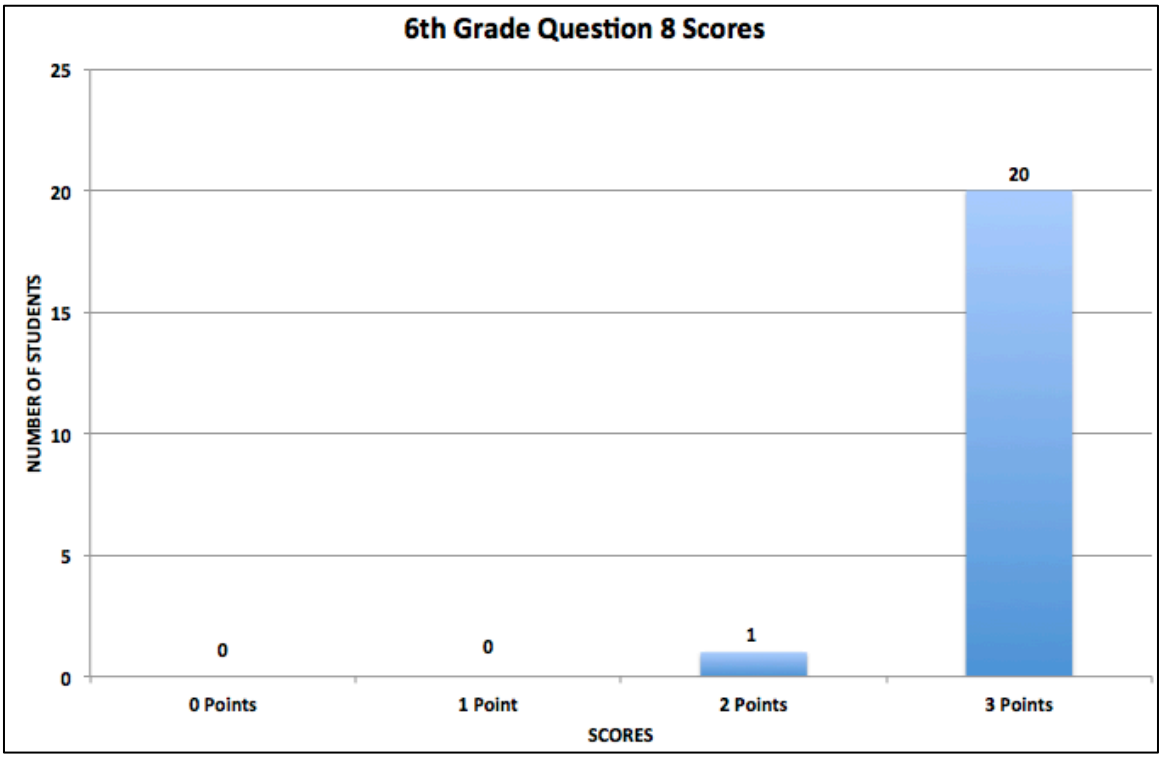


Figure 106: 6th Grade Question 8 Scores

It is hard to say exactly why the 6th graders would do better on this question than the 7th graders as the 6th graders did not get the same classroom instruction involving decomposition. One reason might be the self selected idea talked about in the last section; namely the sixth grade students who returned their forms might not be representative of these two classes. It could be that a few 7th graders might have overthought this question. It is hard to say exactly why this might be the case; however, most students in both classes got the question correct.

The following quotes are typical of correct responses to this question.

"In real life it would take a lot longer to decompose than this simulation."

"In real life, the fox and rabbit would actually take a lot longer to decompose. For example, it takes about the same amount of time for the animals to get hungry and for them to decompose."

Both of these answers talk about the length of time decomposition takes. The second answer even uses the relative simulation time to show how quickly the agents decompose.

Incorrect responses seemed to miss the idea behind decomposition in this question or were simply off base. For example, the following are typical incorrect responses to this question.

"Because it takes a while to die in real life."

"It is realistic"

The top answer might be on the right track if the student replaced "decompose" with "die." The second answer is wrong and furthermore, does not give a reason for why it is realistic. It could be that this student quickly went through the questions in order to continue with the simulation without thinking about the answer. Since I was collecting the worksheets, students unfortunately knew that their grade did not depend on anything they wrote; answers like the second one above were typical of students I noticed trying to finish all the steps of the worksheet as quickly as possible.

Question 9: Question 9 was ill-posed question and thus was not analyzed (see section 4.2.3).

Question 10: Question 10 asks what the Hungry Fox and the Hungry Rabbit Agents still need to do in the simulation and what pattern one would use to achieve this. At this point the Hungry Fox and Hungry Rabbit track their food but they do not eat it. The correct answer would mention a pattern that could be used for eating like change to a dead agent or absorb the agent. The only way to achieve this using the Simulation Creation Toolkit is to have the agent be changed to a dead agent and the dead agent change the hungry

agent back to a regular agent. However, if we did not need to have this change from hungry agent back to regular agent, the Absorb Pattern would work just as well.

The following graph depicts the breakdown of 7th grade students' answers to Question 10.

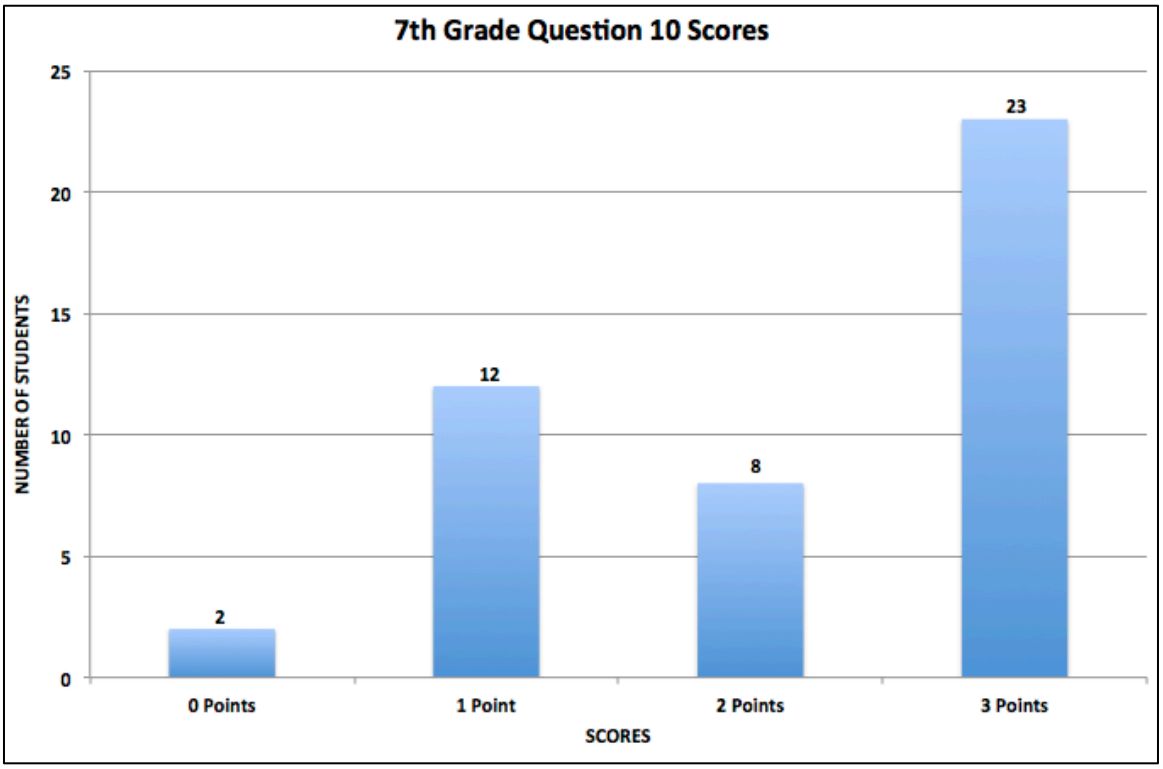


Figure 107: 7th Grade Question 10 Scores

Many 7th graders were able to reason out that the agents still needed to eat and change back into their regular versions. The students who got 2 points had the right idea but did not provide the pattern necessary to accomplish this. Some students claimed that these agents needed to Mate and stated that the Generate Pattern would allow for this. This is correct but is not in the context of the question and thus they got 1 point.

The following graph depicts the breakdown of 6th grade students' answers to Question 10.

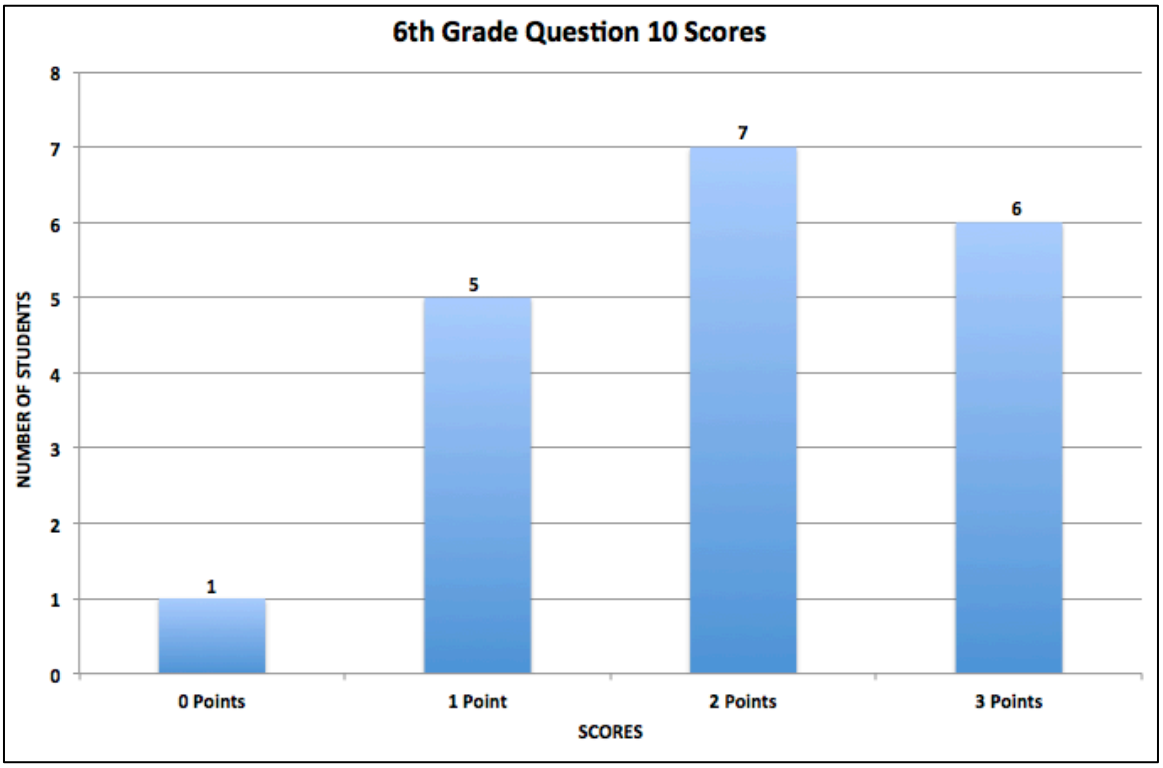


Figure 108: 6th Grade Question 10 Scores

Similar to the 7th graders, many 6th graders forgot to put what pattern they would use to accomplish this interaction. Furthermore, many 6th grade students also put that the hungry agents needed to mate and provided the Generation Pattern as the pattern they would use to accomplish this interaction.

The following quotes were typical of answers deemed correct.

"The fox needs to eat the rabbit by using the change pattern."

"The hungry rabbit and hungry fox still need to become not hungry after they eat. We would use the collision pattern."

Both of these answers mention what the agents need to do and what pattern they would use to accomplish it. The top one specifically states the Change Pattern while the second answer mentions the general Collision Pattern. Examples of the typical answer that got 1 point are as follows.

"Reproduction, one agent creates another"

"They need to reproduce, you could use the generation pattern."

It is puzzling as to why students thought that reproduction was the correct answer to this question given the phrasing of the question itself. One idea is that they thought the hunger issue was taken care of by the agent tracking their food. If a student did not run the simulation to see what was going on or did not completely understand the patterns they were implementing (which is likely given how highly scaffolded this unit is) they might reason that the obvious next step in making this simulation realistic involves adding reproduction.

Question 11: Question 11 asks the students what depictions of Rabbit Agents the Fox Agent eats, what is wrong with the depictions the Fox Agent eats, and finally, how one might change the simulation to make it more correct. Initially, the correct answer was thought to be the Dead Rabbit Depiction for if a Rabbit Agent is already dead, it probably would not get eaten by the Fox Agent. Furthermore, if a Hungry Fox Agent eats a Rabbit Agent, the Dead Rabbit Agent could sustain another Hungry Fox the way the simulation was created. The fix for this issue would be to disallow the Hungry Fox Agent to change an already Dead Rabbit Agent into a Dead Rabbit Agent and/or find a way that the an already Hungry Fox Agent is not changed into a Regular Fox Agent by a Dead Rabbit Agent that has been eaten by another Fox Agent. This is nontrivial using the Simulation Creation

Toolkit as there is no mechanism to ensure a Dead Rabbit Agent only feeds one Fox (i.e. one could have a regular Fox change the Dead Rabbit Agent into another depiction of the Dead Rabbit Agent but that might yield a regular Fox Agent that was never hungry changing the Dead Rabbit Agent). However, many astute students pointed out during the class that the Fox Agent could eat a Dead Rabbit Agent especially if it was about to die and that one Rabbit could be shared among multiple Foxes. Therefore, given adequate reasoning, both of these answers are deemed correct. As mentioned in section 4.3.2, it is hard to draw any conclusions from either of these answers.

The following graph depicts the breakdown of 7th grade students' answers to Question 11.

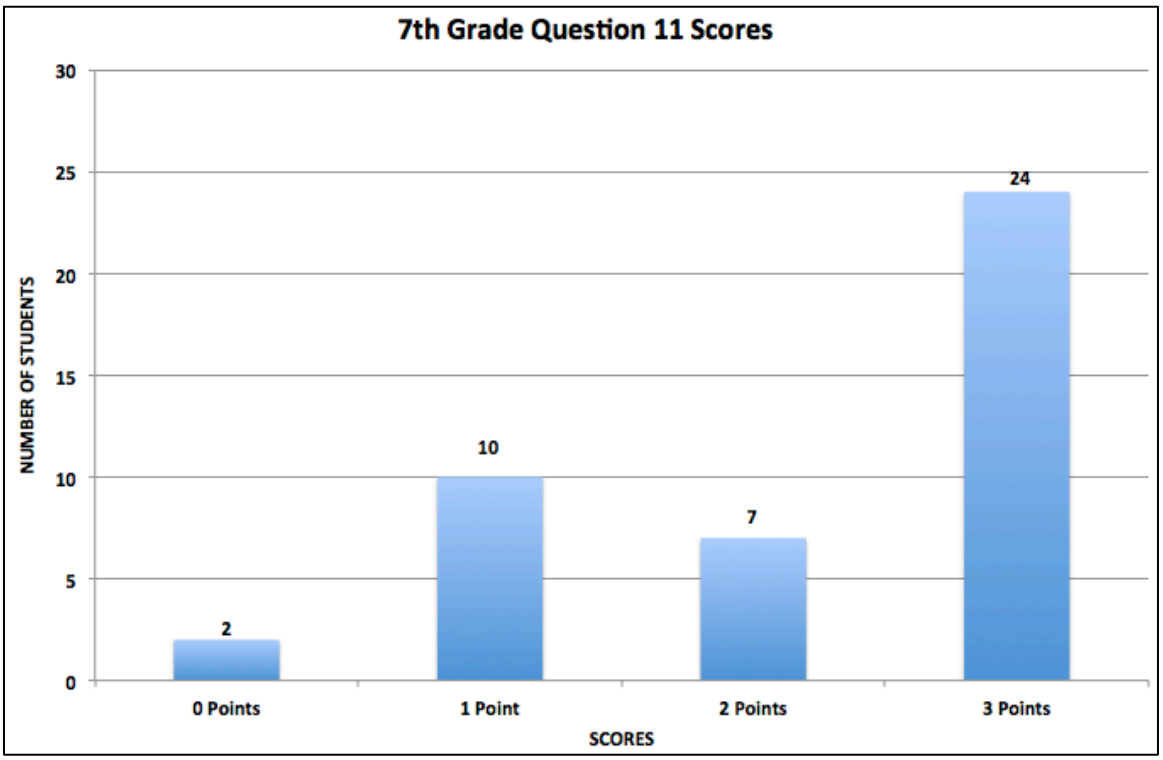


Figure 109: 7th Grade Question 11 Scores

The answers to this question are hard to categorize given the variety of answers students could provide. 1 point meant that the student did not add any additional information other than realistic or unrealistic. Otherwise the student got 2 points for a correct answer with a reason and 3 points with a correct answer, a reason, and how they might change the simulation (or without this information if they thought it was realistic).

The following graph depicts the breakdown of 6th grade students' answers to Question 11.

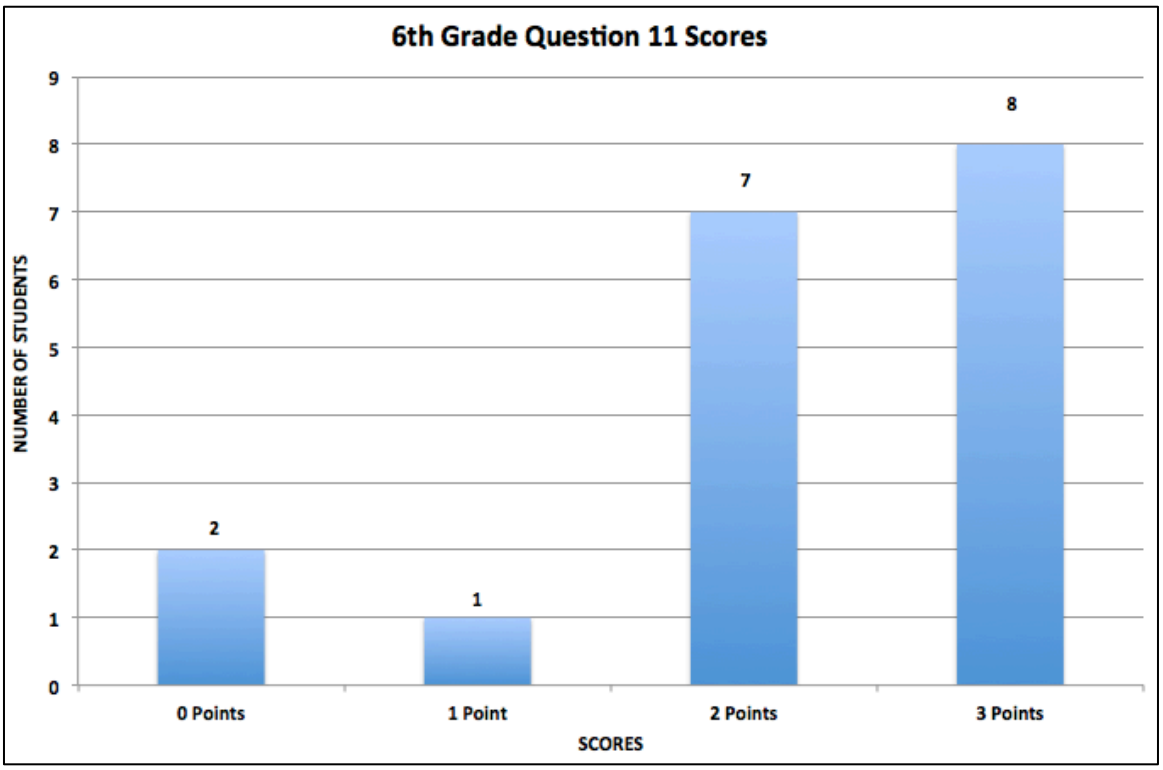


Figure 110: 6th Grade Question 11 Scores

Like 7th grade students, most 6th graders got this question right. Again given the ambiguity surrounding the correct answer, it is hard to draw any conclusions from this.

The following are typical correct student responses to this question.

"A fox could also eat a dead rabbit. If it is not realistic you could program it so that a fox only eats regular and hungry rabbits."

"The dead rabbit might get eaten and it isn't realistic because foxes don't eat dead rabbits. You could change it so that you could only have hungry and normal rabbits get eaten."

"Dead rabbit, yes because the hungry fox might even eat the dead rabbit to stay alive. Yes, you could make it take more time for the rabbit to decompose."

As alluded to in Question 11, the thought was that these answers could be interesting. However, given the array of answers it is hard to gain any insight to this question other than there is evidence that students do understand what might make a simulation realistic.

Question 12: This question asks students to run the simulation and see how long it takes for all the animals to die out. This question is asked more as a motivating question for adding mating to the simulation and thus the answers are not indicative of much.

Question 13: This question asks students what pattern they would use to add mating to the simulation. The correct answer is the Generation Pattern or a description of the Generation Pattern.

The following graph depicts the breakdown of 7th grade students' answers to Question 11.

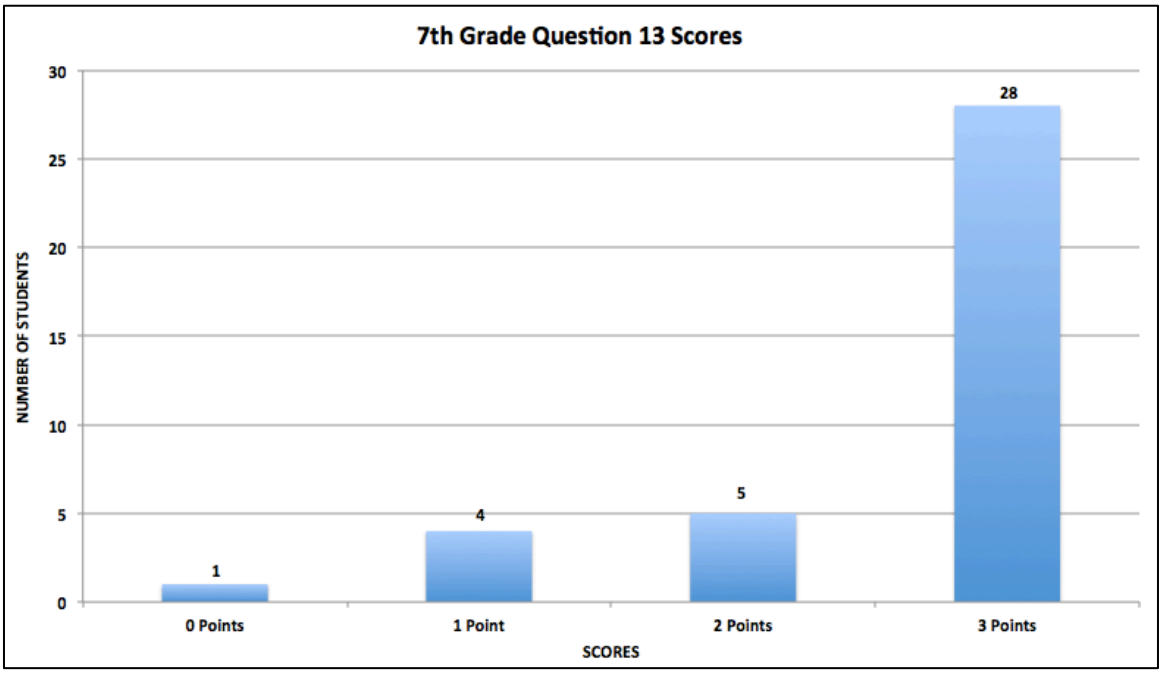


Figure 111: 7th Grade Question 13 Scores

The above figure shows that most 7th grade students were able to get the Generation Pattern. The significance of this question is that if students are able to identify the pattern they would use to create an interaction using the Simulation Creation Toolkit they are more likely to be successful in creating simulations using the toolkit. Furthermore, since this exercise is heavily scaffolded, these questions give insight into whether students can use the tool once step by step instructions are removed.

The following graph depicts the breakdown of 6th grade students' answers to Question 11.

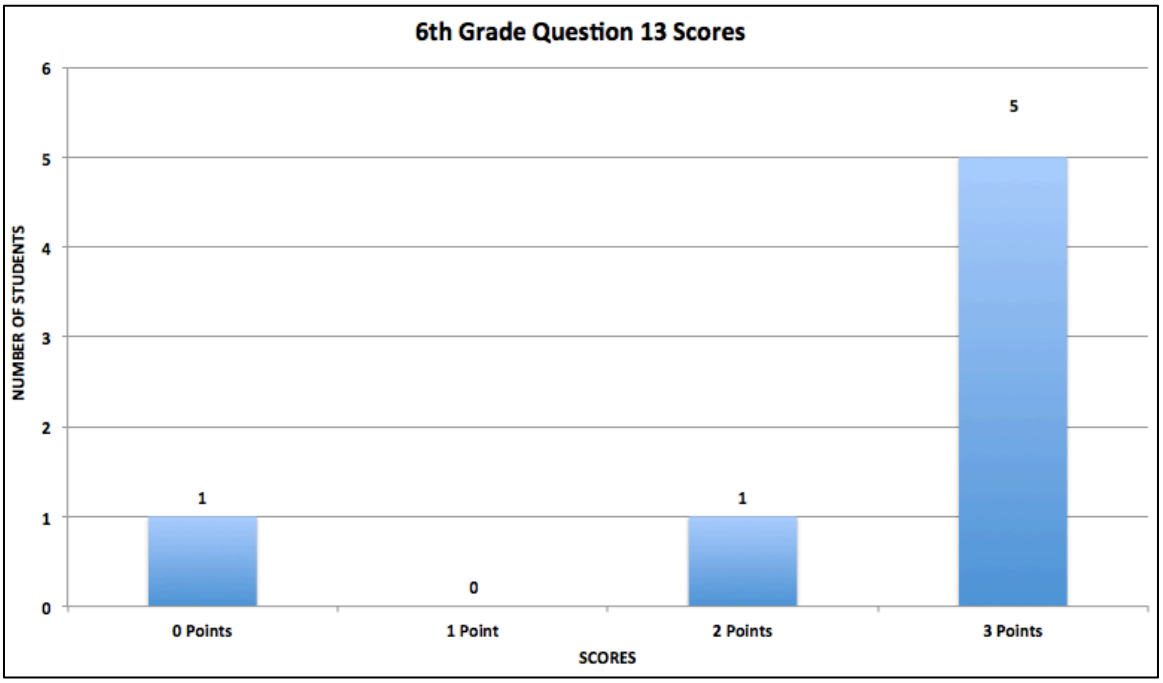


Figure 112: 6th Grade Question 13 Scores

The drop-off in 6th graders answering this question is noticeable from the earlier questions. Nevertheless, of the 7 6th graders who answered this question, 5 of them were able to get it correct.

At this point of the simulation students have seen the Simulation Creation Toolkit for multiple class periods over multiple days; therefore, if a vast majority of 6th and 7th graders were not able to identify the correct pattern at this point then there would be little reason to expect this tool to be effective without specific direction or a tutorial.

The following quotes are typical of correct answers for Question 13.

"The generation pattern."

"Generation would be used to add mating."

"Generation: One Agent creates another agent."

Sometimes instead of mentioning generation, the student would use the description of the pattern. The following is typical of a partial credit response.

"You would use the one where an agent makes another agent."

Questions 14-21, 25: Questions 14-21 and 25 will be grouped together for this section as they all deal with the same basic experiment (see section 4.3.2). It should be noted that since the 6th graders had one less day, only a few 6th grade students got to these questions so the data will focus on the 7th grade class. The "correctness" for these questions is hard to quantify. These questions ask students to run the simulation and see how long it takes for a given agent (Rabbit or Foxes) to die out. Then it asks the students to change a parameter value by altering one of the pattern specifications to try to extend the time it takes for a given agent to die off as well as explain why they think this change will help the prolong the time it takes for an agent to die off. Students could alter the hunger rate, the death rate, the mating rate, animal speed among other possible choices. It should be noted that Fox Agents die off first with no changes to the simulation pattern parameters if the simulation is completed entirely correct.

These questions were meant as a way to have students change pattern specifications to run a small experiment. It was supposed to be a proof of concept that the Simulation Creation Toolkit could be used to run quick experiments after the patterns were implemented constituting a simulation. As mentioned in section 4.3.2, often students used unrealistic values for their simulation parameters. For example, students set the percentage pertaining to reproduction to 100% which meant that every time a Rabbit Agent or Fox Agent would come into contact with another Rabbit or Fox Agent, they would reproduce. This leads to unconstrained population growth for these agents meaning they will never die out but is not a realistic parameter for an ecosystem. Similarly, students could set the hunger rate (the specification corresponding to the Dirt Agent changing the regular Fox Agent into a Hungry Fox Agent) extremely low, for example. This leads to

regular Fox Agents never becoming Hungry Fox Agents and thus less Foxes dying.

This idea relates to the general theme of unrealistic simulations possibly yielding misconceptions. Subsequent questions ask students how hard it is to keep a population in equilibrium such that every Fox that dies is replaced by another Fox Agent and/or every Rabbit Agent that dies is replaced by another Rabbit Agent (this is a simplification of the idea of equilibrium but given the time constraints worked for this experiment). If a student was to set the hunger rate to zero and the mating rate to zero, the populations of animals would never get hungry and never mate making equilibrium somewhat easy to achieve. A better way to present these questions would have been to constrain each parameter such that students could only put in a range of parameter values.

The following graph depicts the breakdown of 7th grade students' responses to Question 17 wherein students are asked to give a reason why changing the specification parameter they chose to modify will increase the time it takes for the first animal to die out.

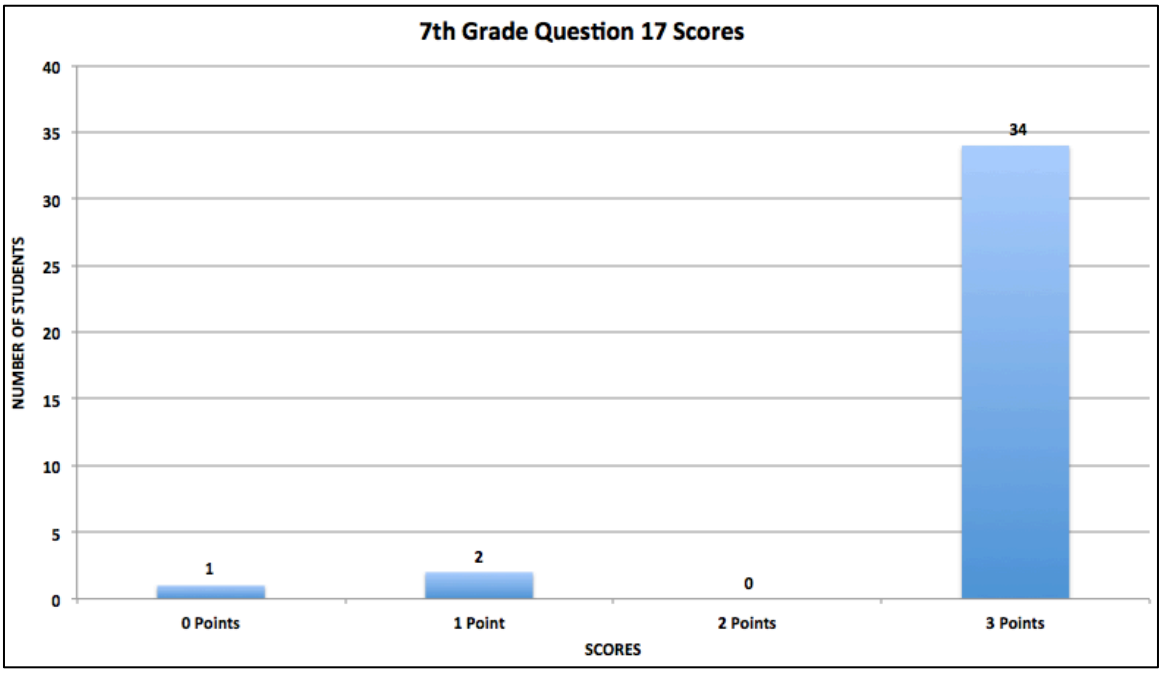


Figure 113: 7th Grade Question 17 Scores

Though these experiment questions are hard to quantify, the above figure shows that most students had reasonable logic as to why changing a particular parameter value might yield an increase in population survival. In general, students were able to understand the connection between changing parameters and the effect they might have on the simulation. Furthermore, students could form a hypothesis and use the Simulation Creation Toolkit to implement their hypothesis using the simulation parameters.

Question 18 asked students to run the simulation and record their results. Given that students changed a given parameter correctly and their simulation was built such that they got sensible results, this question was deemed correct. It should be noted that this does not necessarily mean that their simulation was implemented perfectly; just that their results were reasonable given the parameter change they did. The following graph depicts the breakdown of 7th grade students' responses to Question 18.

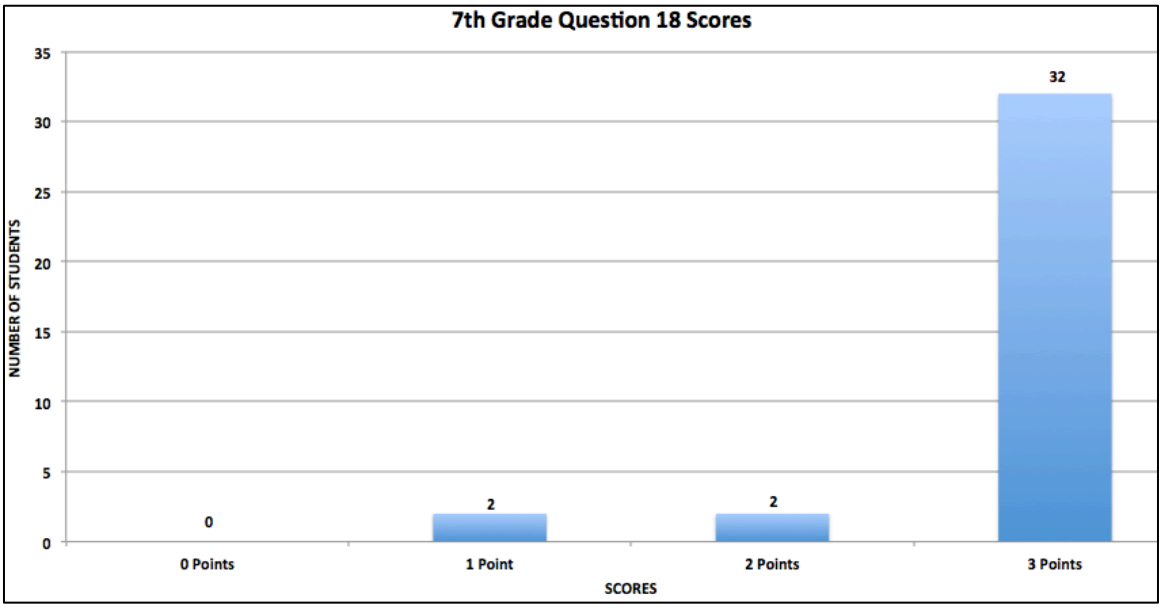


Figure 114: 7th Grade Question 18 Scores

The above graph shows that most students actually obtained a reasonable result for their parameter change. An example of a reasonable result would be if a student increased the mating rate of the Foxes to 100% and the Foxes overran the Rabbit Agents yielding no Rabbit Agents over time. When viewed in concert with the Question 19 scores, wherein students attempt to explain the results they obtained, the results for this mini experiment look promising. The following graph depicts the breakdown of 7th grade students' responses to Question 19.

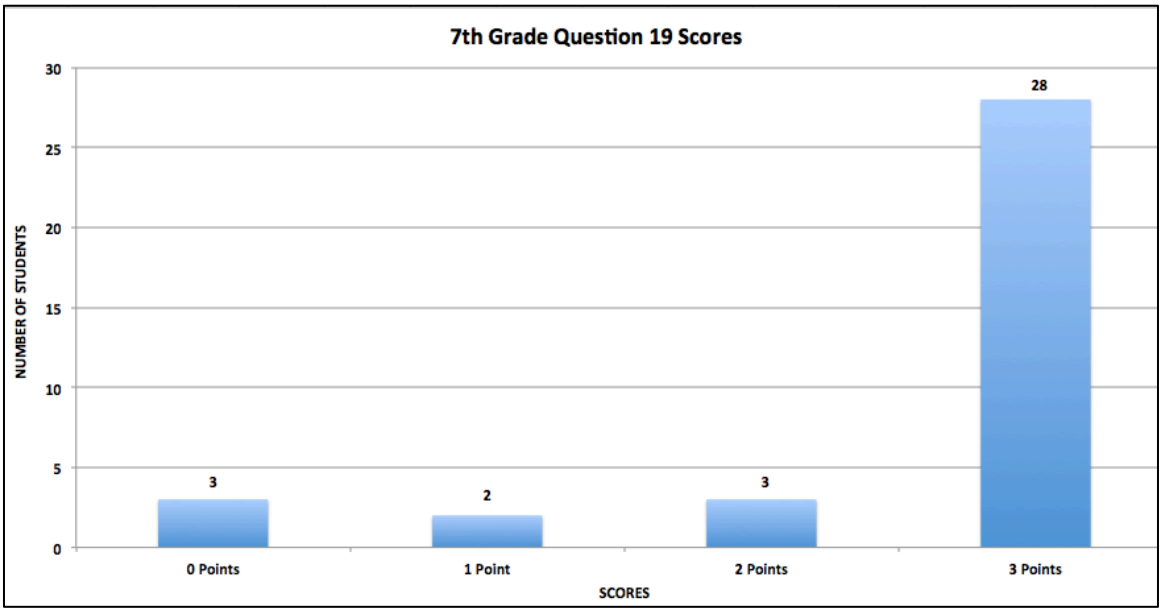


Figure 115: 7th Grade Question 19 Scores

The answer for Figure 19 was marked correct if it contained an explanation of the results obtained in Figure 18 which made sense given the implemented parameter change for this part of the unit. As with Question 18, most students got this answer correct too, meaning they were able to use their simulation to make a hypothesis, give a reason why the hypothesis would work, implement the hypothesis by changing a parameter, run their simulation such that they obtained a reasonable result given their parameter change, and give a plausible explanation as to why they obtained a given result using the Simulation Creation Toolkit. More studies wherein students run in-depth experiments on their system should be tried before making any concrete conclusions about the extent students can use this system to create and experiment on simulations, but as an initial data point, the students' answers to these questions seem promising.

The final question of the worksheet (Question 25) asks students to add a Poacher Agent that eliminates Fox Agents and asks what effects the Poacher Agent has on population equilibrium. Students received much less

guidance in implementing this question. Students had to figure out how to implement 3 patterns—Poacher Agent Generating a Bullet Agent, the Bullet Agent moving through Directional Movement, and finally either the Bullet Agent changing the Fox Agent to a Dead Fox Agent or Absorbing the Fox Agent. 27 students were able to complete the patterns necessary to create a Poacher Agent. If students had a reasonable answer to this question, taking into account the state of their simulation post parameter change, it was deemed correct. The following graph depicts the breakdown of 7th grade students' responses to Question 25.

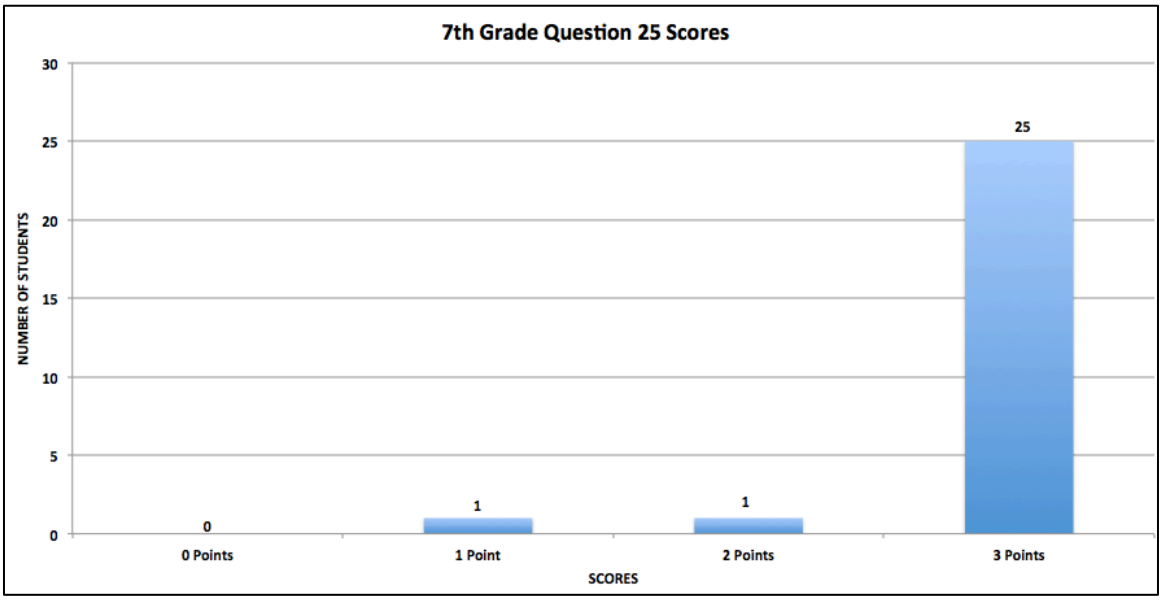


Figure 116: 7th Grade Question 25 Scores

The above figure shows that almost all seventh graders who got to this point had a reasonable answer to Question 25. Interestingly enough many students who had set the mating rate of the Generation Pattern specification extremely high stated that Poachers actually helped the system stay in equilibrium because they eliminated the unconstrained growth of the Fox Agent population. In this situation, the introduction of Poacher Agents would

actually help limit the population. The Guaranteed Viable Curriculum asks that students learn that humans have impacts on ecosystem populations both directly and indirectly. This is an example of direct population impacts of humans, however it is a positive impact which is not necessarily an aim of the curriculum.

The following is a typical sequence of a student carrying out the above experiment correctly.

The Parameter Changed: *"The percent 80% chance of foxes mating."*

Reason Why This Change Will Increase Simulation Time:*"I think it will increase the time for foxes to die off because mating is more likely."*

Results Of The Experiment: *"The rabbits died off before the foxes, it was shorter by 13 seconds."*

Reason For The Results: *"Because the foxes breed more often."*

Is This System Hard Or Easy To Keep In Equilibrium: *"It is hard to keep in equilibrium because each has to be breeding at the same rate."*

After Introducing Poacher Agents, Is This System Harder Or Easier To Keep In Equilibrium: *"It made it easier to keep the system in equilibrium because the foxes didn't reproduce too quickly."*

The above student allows the foxes to mate 80% of the time when one Fox Agent comes into contact with another Fox Agent. The student hypothesizes that this will increase the time of the simulation because Fox Agents will not die out as fast. However, the increase in Foxes leads to Rabbit Agents dying out faster and the time actually decreasing by 13 seconds. Given this high mating rate the student surmises that the system is hard to keep in equilibrium and finally, the student writes that the addition of Poacher Agents actually helps keep the system in equilibrium. It should be noted that the student could have changed other parameters in an attempt to keep the

system in equilibrium; this quick experiment is not ideal as it leads students to put unrealistic parameter values to make the populations stay the same or draw conclusions based on previously implemented unrealistic parameter values.

The following is another typical sequence carried out by students.

The Parameter Changed: *"I drastically changed the percent that dirt changes a hungry rabbit to a dead rabbit."*

Reason Why This Change Will Increase Simulation Time: *"The rabbits will not die so their population will go up and it will take longer for them to die."*

Results Of The Experiment: *"It increased by a lot…"*

Reason For The Results: *"The rabbits only died .1 percent of the time."*

Is This System Hard Or Easy To Keep In Equilibrium: *"It would be easy once you reached equilibrium to keep it there, because there would be no sudden disasters that harm a population."*

After Introducing Poacher Agents, Is This System Harder Or Easier To Keep In Equilibrium: *"It is harder with the poacher to keep equilibrium because he suddenly kills more than is natural."*

This student decided to tweak the death rates for the rabbit. Therefore, rabbits would get hungry but rarely die. With many rabbits to eat, the Foxes were less likely to die and the simulation time increased (neither agents probably ever died off). This strategy, though different from the first strategy, is a way to indirectly increase the time it takes for the Fox Agent population to die off completely, and since the Fox Agent population does not over-run the Rabbit Agent population, the simulation time should increase. This student concluded that once the system was put in equilibrium it would be easy to keep it there because, unlike the real world, there would not be any external forces that could effect populations. Comparing this constrained

simulation to the real world, though unexpected, this is a logical argument. In this case the Poacher kills the Fox Agent. Given enough Poacher Agents the Fox Agents would be eradicated fairly quickly (also taking into account that Fox Agents cannot hide in the AgentCubes world and the Poachers constantly shoot Bullet Agents). Unlike the last example, the foxes in this student's simulation do not constantly create new Fox Agents and therefore, the Poacher Agents effect the Fox Agent population adversely.

The following is a final example of a student carrying out the above experiment.

The Parameter Changed: *"1% chance of a rabbit reproducing."*

Reason Why This Change Will Increase Simulation Time: *"I think the time will decrease."*

Results Of The Experiment: *"decreased"*

Reason For The Results: "*I think I got this result because rabbits aren't reproducing and dying off quickly."*

Is This System Hard Or Easy To Keep In Equilibrium: (after changing the fox chance of dying and rabbit chance of dying to 2% and 1% respectively) *"it is very hard to keep the system in equilibrium."*

After Introducing Poacher Agents, Is This System Harder Or Easier To Keep In Equilibrium: (after changing the fox chance of dying and rabbit chance of dying to 2% and 1% respectively) *"It makes it easier to keep the system in equilibrium."*

This student decided to go against the challenge and actually do something that decreased the time for one of the animal agents to die out (as noted in the responses, the student was well aware of this). Specifically, this student changed the percent chance associated with Rabbit Agent reproduction to 1%. With less Rabbit Agents reproducing, Fox Agents can

consume the Rabbit Agents fairly quickly. The student hypothesized that this would decrease the time it took for one of the animal agents to die out and, in fact, this did occur. The student then changes the chances of Rabbit Agents and Fox Agents from dying to 2% and 1%. Thus Rabbit Agents are not reproducing and Foxes and Rabbit Agents are not dying. Foxes, however are reproducing (and at some point the reproduction will become unconstrained as there is a limited amount of space in the simulation world for Foxes to occupy leading to more Fox Agents next to other Fox Agents). This student states, much like the first student who increased Fox Agent mating to 80%, that this system is very hard to keep in equilibrium, and further, states that the inclusion of Poacher Agents makes the system easier to keep in equilibrium. This is probably because the Poacher Agents negate the unconstrained reproduction of Fox Agents.

Questions 22,23,24: Questions 22, 23 and 24 present students with a depiction of a trophic pyramid of biomass at each level of an ecosystem. Since foxes are higher up on the pyramid than rabbits, one would think that our simulation should have less biomass at the Fox Agent level than the Rabbit Agent level. However, the initial setup for the experiment has 36 Fox Agents and 54 Rabbit Agents. Assuming 15 pounds for each Fox Agent and 4 pounds for each Rabbit Agent (these figures are within the ranges for fox and rabbit weights) we get 540 pounds of Foxes and 216 pounds of Rabbit biomass. Question 24 asks students if this is realistic and how might they change this simulation if it is not. The correct answer should make reference to the fact that less Fox Agents or more Rabbit Agents are needed in the simulation to make the Rabbit Agent biomass exceed the Fox Agent biomass.

The following graph depicts the breakdown of 7th grade students' answers to Question 24.

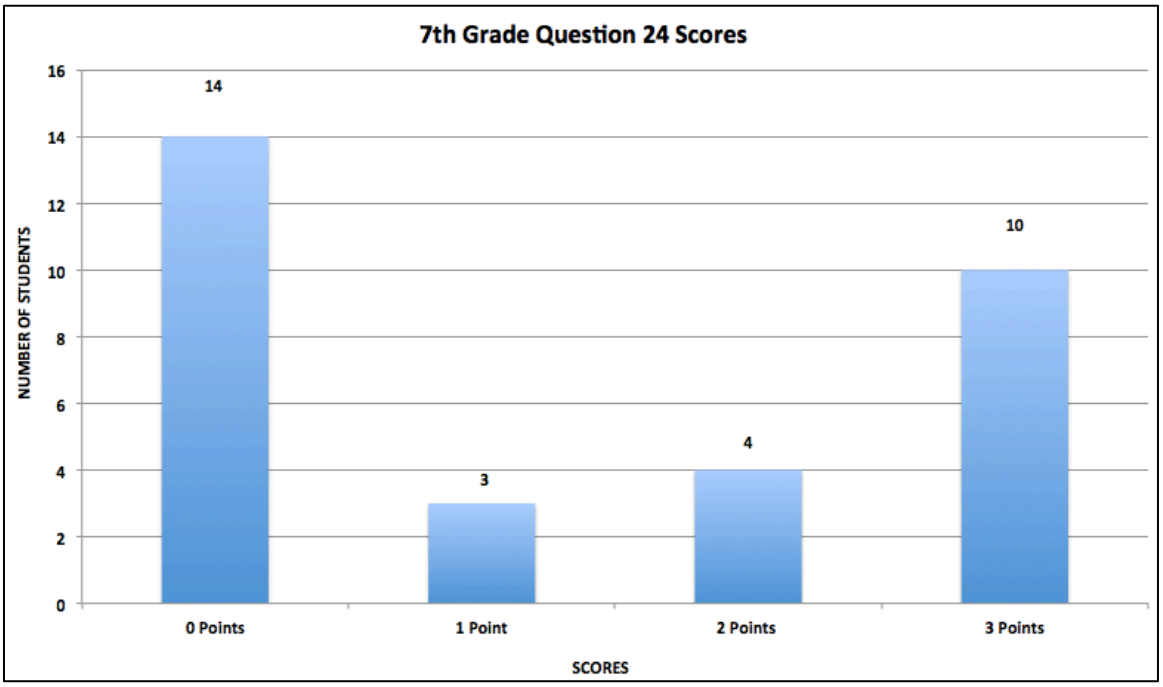


Figure 117: 7th Grade Question 24 Scores

Almost every seventh grade student got the answer to Questions 22 and 23 correct; therefore, it was surprising to see how many students missed the concept behind Question 24. The answers students often gave were that this simulations was realistic in terms of the trophic pyramid because Fox Agents eat Rabbit Agents or Rabbit Agents decompose and thus the simulation was realistic. Basically, students discarded the idea of biomass and the trophic pyramid and talked about other factors of the simulation that could be perceived as realistic. It is still not completely clear why this is the case; it could be the wording was confusing – instead of saying based on the pyramid above the question might have made more sense if it explicitly referenced the biomass calculations students did in the preceding questions.

The following quotes are typical of correct responses to this question.

"No, I would make less foxes."

"Our simulation is not realistic. I would change it by starting out with less foxes."

Some students pointed out other parts of the trophic pyramid that were incorrectly implemented in this simulation. These students got partial credit. The following quotes are typical of these answers.

"No, because the size of an organism is not relevant in the simulation. We could change it by mating weight factor."

"No, we could add primary predators."

Both of these answers are correct, but not necessarily the answers expected given the calculations completed before. These answers imply that the phrasing of the question should be changed to reflect that the biomass calculations are to be used in answering this question. The following quotes are typical of answers that were deemed incorrect (1 or 0 points):

"Yes because they eat and then decompose."

"Yes it is realistic."

The first answer mentions an aspect of the simulation that is correct and uses it as a reason for why this simulation is realistic taking the trophic pyramid into account. Namely, this student probably saw that decomposers were included in the pyramid, and noted that the agents decompose into grass in the simulation concluding that the simulation was realistic. The second answer does not give a reason for why the simulation might be realistic as compared to the trophic level pyramid but many students pointed out the existence of Foxes and Rabbits in the simulation or the fact that Foxes ate the Rabbits which was realistic.

5.3 Relationship Between Simulation Results and Worksheet Results

As students implement their Predator/Prey simulation they answer worksheet questions. Many of these questions relate to the simulation they have created up to this point or patterns they will subsequently implement to complete the simulation. This section will look at the results of the simulation creation activity results and how it relates to the worksheet question results.

5.3.1 Results Of Simulation Results And How They Relate To Worksheet Results

A review of how worksheet questions related to how well students programmed the simulation up to the point the question was asked showed there was no significant correlation between the two activities. It was hoped that this study would show some correlation between the two; however, there are many reasons why this is the case. This section will discuss why the results show no correlation.

The first issue is that the questions in the worksheet were not written in such a way that running the simulation would necessarily help students deduce the correct answer. The worksheet questions were written such that the not only fit into the context of the simulation, but, as importantly, reinforce often-misunderstood concepts present in Centenary Middle School's Guaranteed Viable Curriculum. It was initially thought that these two goals could coexist with the overarching goal of seeing if the simulation creation activity itself could facilitate student understanding. However, to measure this in part necessitates questions that one could arguably not have answered before the simulation creation exercise but can answer after the exercise is completed. Phase 2 in section 4.2.3 looks at each

question in this context and concludes that most if not all of these questions do not fit into this category.

The concepts outlined in the Guaranteed Viable Curriculum for ecosystems do not easily lend themselves to simulation creation. For example, for the purposes of the GVC, students should learn that "decomposition is the breakdown of organic material." The GVC also states that students should learn "infers the number of organisms or amount of energy available at each level of an energy pyramid." Building a simulation wherein decomposition is included as part of an ecosystem is more in-depth than what the GVC has students learn. Similarly, having a question wherein students look at the number of organisms at each level of the energy pyramid in the simulation might reinforce the concept but the simulation itself is not necessary for a concept like this. In fact, many items in the Life Science GVC are concepts that can be tested through knowledge assessment; they consist of facts that students should know. Though important for other reasons that will be covered, basing worksheet questions to accommodate the Guaranteed Viable Curriculum might not have been the best strategy for seeing if simulation creation actually helps student understanding.

The Guaranteed Viable Curriculum does include parts wherein students make hypotheses and collect and analyze data. Instead of hitting a large amount of GVC concepts in the limited time, a better strategy might have been to hit this specific part of the GVC as it involves actually using the simulation students created. Estimating the time from the epidemiology unit, this probably would have taken one and half periods to allow students to run trials and analyze their results. Therefore, the simulation creation exercise might have had to be shortened; however, this could be done by simplifying the Predator/Prey simulation that these middle school students created.

Having students collect, record, and analyze trends in data focusing on concepts such as what occurs when simulation parameters are changed, would enable the simulation creation act itself to be essential to the worksheet activity. Furthermore, the student understanding would not be limited to the subject of the simulation, in this case ecosystems, but would focus on the more overarching points of the Guaranteed Viable Curriculum wherein students deal with models and data analysis.

For this study, the focus of this tool was not necessarily to create realistic simulations. Though realistic simulations is an important part of any modeling activity, it is up to the class instructor to decide what level of realism is necessary for a given modeling exercise. This thesis was an attempt to use Computational Thinking Patterns as a strategy to more quickly and easily build simulations. To this end the Simulation Creation Toolkit provided students with a palette of patterns that allowed students to create simulations using groups of conditions and actions as the primary units of simulation construction. This proof of concept study has some agent interactions that could be viewed as realistic (i.e. the Hungry Fox pursuing and eating the Rabbit Agent) and some interactions that might not be (i.e. on average every few steps the Fox Agent becomes hungry). Students are really building a system of agents that interact with each other. By changing the parameters the students can make such a system more or less realistic depending on what they are specifically modeling. To make the Simulation Creation Toolkit handle an array of simulations this freedom with specifications is essential (i.e. there might be a modeling exercise that requires a high mating rate for example).

Given that this is the case, this study wherein students put together a Predator/Prey simulation that is not necessarily realistic, makes using the

simulation itself to draw any conclusions by running the simulation such that they fit the Guaranteed Viable Curriculum is a tough task. For example, the GVC states that students should be able to "describe the impact of humans on the environment and how that affects the survival of populations and entire species." However, given an unrealistic parameter of the Generation Pattern that enables Fox Agents to reproduce constantly, one might logically find that a Poacher Agent actually helps the unconstrained growth of the Fox Agent. Furthermore, in the simulation, Fox populations can grow unconstrained with the only limiting factor of their growth being death by hunger if no Rabbit Agents exist. However, since Foxes take a while to die from hunger in the simulation, they can mate before they die. Therefore, a student might conclude by running the simulation that the population of Rabbits and Foxes in the simulation are not intertwined, and given their particular parameter values for their simulation, this conclusion is correct. In terms of actually increasing student understanding of real world phenomena this is counter productive.

Though there is no correlation between the simulation creation parts of the unit and performance on the worksheet, there is value in this simulation creation activity in terms of the classroom environment. For one, as pointed out in Chapter 1, there are very few modeling and simulation creation activities in the middle school classroom today. However teachers and policy makers believe that these activities are important to include in the curriculum. However, the concepts students must learn according to the Life Science GVC at Centenary Middle School, for example, are not items that a simulation creation unit would necessarily facilitate. Therefore, there is no advantage for a teacher to integrate these types of units into their classroom. Furthermore, the few GVC items that a modeling activity could be used for—

creating hypotheses, collecting data, and analyzing data, can all be done within any physical lab setup and can be met within one week of class time. Using a modeling activity to cover this one concept would not necessarily lead to simulation creation activities to be integrated throughout the duration of the class.

Having this unit hit upon numerous GVC concepts, though unfocused, yields a few advantages. First, the unit can take up more class time as more essential curriculum items are covered during this period of time. Secondly, it enables these modeling activities to coexist within the current GVC. Having this activity cover many of these GVC items in the context of simulation creation is a way to get simulations into the classroom without having to wait for an updated GVC that may or may not add greater emphasis to modeling activities in the classroom. In this way, modeling activities could be used throughout the semester with students creating various simulations and answering questions within the context of these simulations that reinforce essential concepts outlined in the GVC.

5.3.2 Ability of Students To Connect Simulation Creation Toolkit Patterns To Simulation Behavior

Some of the worksheet questions ask students how they would implement a given interaction in the simulation. Since students are given step by step instructions on how to create the simulation using the Simulation Creation Toolkit, these questions help provide insight into whether students actually understand what pattern to use to implement an interaction or if they are relying on the worksheet instructions to create the simulation without truly understanding what the patterns accomplish.

The questions that enable students to name or describe a pattern they might use to implement a given interaction are Questions 4, 7, 10, and

13. By combining the results of these four questions we can get a better idea as to how students responded to the Simulation Creation Toolkit itself. The following graph depicts all the 7th grade students' answers to these four questions and the percentage of total answers in each scoring category.

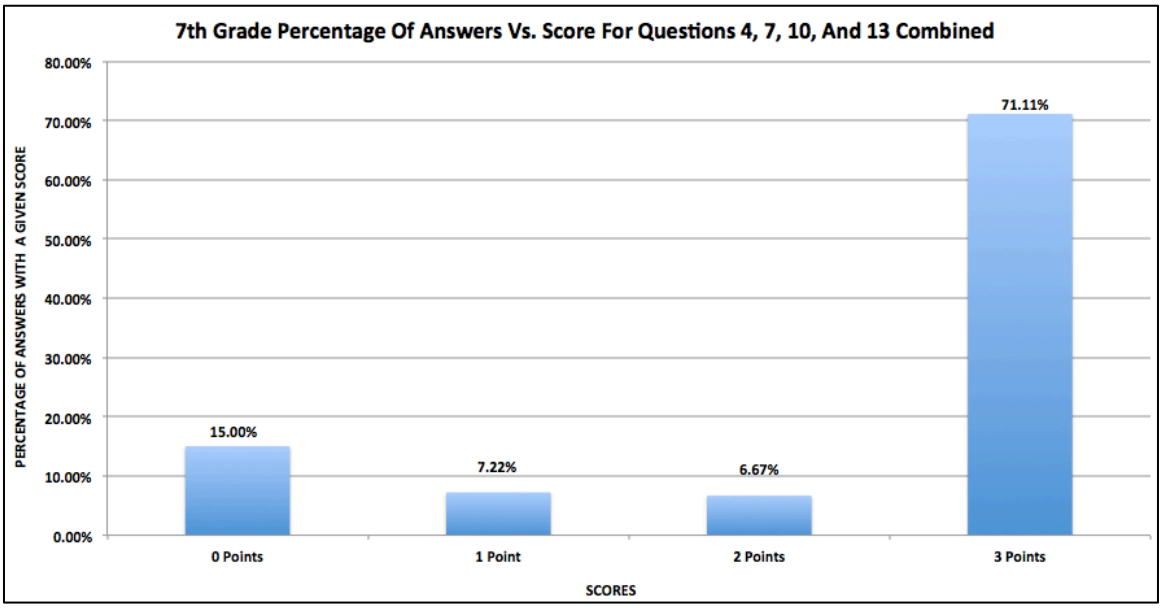


Figure 118: 7th Grade Percentage Of Answers Vs. Score For Questions 4, 7, 10, And 13 Combined

The above graph shows that 70% of the answers to these four questions were totally correct. In general, the 7th grade students were able to pick the correct pattern using the Simulation Creation Toolkit. However, 70% means that there is ample room for improvement in the system. For example, in general if students are able to figure out the correct pattern to use 70% of the time, in a simulation of 10 patterns on average there would be 3 incorrect. Note that these mistakes would be at the pattern level, not the specification level. Given that a major possible advantage of this system is to enable students to program by analogy, the inconsistency of students to identify the correct

pattern to use without any guidance is a major issue in using the Simulation Creation Toolkit.

The following graph depicts all the 6th grade students' answers to these four questions and the percentage of total answers in each scoring category.

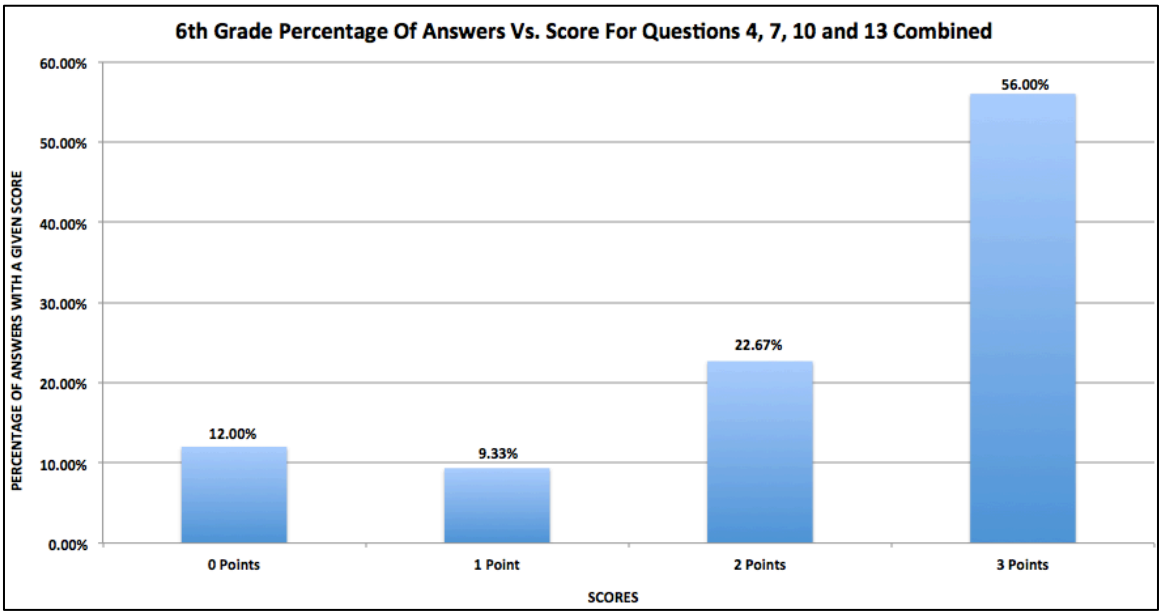


Figure 119: 6th Grade Percentage of Answers Vs. Score For Questions 4, 7, 10, And 13 Combined

The data from the 6th grade students is even more concerning; students were able to get the four questions in this category completely right 56% of the time. Again, for the tool to be successful, one would hope that students would preform better on these types of questions. 6th grade students had little prior exposure to this idea of agent-based modeling. It could be that given more exposure to agent interactions these students scores might increase.

There are a few reasons why the results could be so low. One is that students may not have completely understood all the questions. For example, on question 10 students are asked what the Hungry Fox and Rabbit still need to do in the simulation. At this point of the simulation he Hungry Fox and

Rabbit track their food but they do not eat their food; thus they get hungry and die. If students run the simulation this becomes apparent. However, students who are thinking of the simulation as a whole, there are many things these agents still need to do. As mentioned in the previous section, many students answered that these animals still need to mate which is true but not really what the question is getting at. In fact, many students stated that these agents should mate and the Generation Pattern would accomplish this which is the correct pattern for that particular interaction. The question could have been phrased such that students are better led in the direction of the pattern, however, the question would have had to be phrased as to not lead students directly to the answer. For example, it might have been a better strategy to force students to run the simulation, ask what was incorrect about the animal behavior and then ask them to answer the question. Question 13 is a more straightforward question in this respect. It specifically asks students what pattern would they use to add mating to the simulation. There is much less ambiguity in this particular question; even so, as depicted in the previous section, almost 30% of 7th grade students got this question wrong.

Another possibility is that the user interface of the Simulation Creation Toolkit makes it hard for students to pick the correct pattern. There are two ways students can pick the correct pattern out of the interacticon palette; the student can either interpret the animations or read the accompanying text. How one interprets the animations is highly subjective. Furthermore, according to the middle school teachers, students are notorious for ignoring accompanying text. The text itself gives specific examples of where the pattern might be used. It might be useful to actually display these examples in context when a cursor hoovers over the animation. For example,

rather than having the text corresponding to the Generation Pattern state that this pattern could be used to have tunnels generate trucks, instead have a small animation of the tunnel actually generating the truck among other animations. For each pattern, multiple animations appearing with different actual in-game or simulation examples of what the pattern is capable of doing. This might further solidify the pattern in students minds without over-reliance on text that can easily be disregarded.

Another issue might be the limited exposure students have to the system. Under ideal circumstances, this system would be tested over multiple units. The first unit would involve a heavily guided exercise akin to the Predator/Prey unit. Subsequent units would involve less guidance or no guidance. Given that this is the first unit, many students have never implemented the patterns they are being asked about or explored how they might work in a simulation. For example, at the point the students have to figure out the Generation Pattern for mating, they have never used the system to implement the Generation Pattern and have just learned the concept of patterns a few days prior. It could be that if students had already been exposed to the patterns once before, they would have had a better chance at correctly answering these questions.

5.4 Analogical Reasoning Study Results

Chapter 4 introduces the analogical reasoning study as a way to see how people might use the Simulation Creation Toolkit without step-by-step instructions. To this end 6 participants, 3 with prior exposure to Computational Thinking Patterns and 3 with no prior experience with AgentSheets/AgentCubes or Computational Thinking Patterns, were

provided with three programs that included agents with no behaviors and a pre-made level. None of the participants had ever used the Simulation Creation Toolkit before this study. These participants were also given 3 high-level descriptions of what to create for each program (see Appendix E and Section 4.4). The participants attempted to create a Pacman game, an Epidemiology simulation, and a Predator/Prey simulation.

In addition to the actual simulations data for this section includes notes that were taken while participants were creating their programs and two feedback questions participants answered after creating their final program. As mentioned in Section 4.4 participants were given a short intro to the Simulation Creation Toolkit and then asked to try their best to create the program. Participants were encouraged to talk through what they were trying to accomplish at a given time and were only interrupted if they were about to encounter a system bug or if they asked questions that could not be answered using the Simulation Creation Toolkit (i.e. AgentCubes behavior questions etc.). Finally, participants had as much time as was needed to complete their program; the time duration was recorded. The following sections go through the results of each exercise given to participants followed by the additional results of the long answer questions and note taking activity. The description for each program can be found in Appendix E.2.

The interactions for each simulation were assessed on a 3 point scale. A "√" meant that the interaction existed in the simulation as written in the description. A "√-" meant that the interaction was slightly off from something described. An "X" meant that the interaction was wrong. Since this study was completed on such a small scale, each wrong interaction will be reviewed in the results and discussed in the final discussion section for these results.

5.4.1 Pacman Program Results

The Pacman game participants were asked to program involved the following four interactions:

1) Pacman moves with keyboard keys and cannot move through walls
2) All Ghosts pursue Pacman and cannot move through walls
3) When the Ghosts get to Pacman, Pacman disappears.
4) Pacman eats pellets as he navigates around the level

The following table displays the Analogical Reasoning Study results for the Pacman program; the participants highlighted in yellow refer to people who were involved in the Summer Institute and the participants highlighted in green refer to people who where not.

	Duration	**Inter. 1**	**Inter. 2**	**Inter. 3**	**Inter. 4**
Participant 1	13 minutes	√	√	√	√
Participant 2	19 minutes	√	√	√	√
Participant 4	14 minutes	√	√	√	√
Participant 3	20 minutes	√	√	√	√
Participant 5	36 minutes	√	√	√	√
Participant 6	30 minutes	√	√	√	√

Table 13: Pacman Results For The Analogical Reasoning Study

The results show that every participant was able to create Pacman perfectly based on the description given. Pacman was the easiest exercise probably because, as will be shown with the other two programs, the interactions in Pacman mimicked the interacticons closely. Therefore, it was easier to see which patterns would work for a given interaction.

There were only a few variations on how participants implemented this simulation. Some participants had the Ghost absorb Pacman if it saw Pacman in any direction. Other participants implemented the game such that

Pacman had to be stacked above or below a Ghost to be absorbed. Every participant figured out that the Ghosts must track Pacman and be allowed to move on the Pellet Agents and the Background Agents. If the participant also picked the Pacman agent as a valid agent to move on they were informed of the bug in the tool and told that even without this specification the Ghost agent should automatically move onto Pacman (see section 3.6.6).

The only questionable game, turned in by Participant 4, involved a Pacman, that absorbed pellets in all directions. Therefore as Pacman moved around the level the four pellets around Pacman would disappear. The participant expressed that this interaction was better than the normal Pacman interaction of absorbing 1 pellet at a time and so it was intended. Furthermore, the interaction did not technically violate the description given.

5.4.2 Epidemiology Program Results

The Epidemiology simulation participants were asked to program involved the following four interactions:

1) All depictions of the Person move around randomly
2) Once every second, a normal Person has a 30% chance of becoming sick if next to a Sick Person (depiction of Person Agent).
3) A Sick Person has a 30% chance of recovery once every second.
4) A Sick Person has a 10% chance of death once every second.

The following table displays the Analogical Reasoning Study results for the Epidemiology program; the participants highlighted in yellow refer to people who were involved in the Summer Institute and the participants highlighted in green refer to people who were not.

	Duration	Inter. 1	Inter. 2	Inter. 3	Inter. 4
Participant 1	15 minutes	√	√-	√-	√-
Participant 2	9 minutes	√	√-	√	√
Participant 4	18 minutes	√	√	√	√
Participant 3	13 minutes	√	√	√	√
Participant 5	40 minutes	√	√-	√-	√
Participant 6	44 minutes	√	√	√	√

Table 14: Epidemiology Results For The Analogical Reasoning Study

The above table shows that Participants were able to get the general patterns correct with small deviations from the description. The Epidemiology simulation involved an interaction that was unintuitive for participants to create. Namely, somehow a Sick Person had to get better or die with no obvious interaction with another agent. Participants created these interactions in a few different ways. To make the Sick Person get better it was possible to have the Sick Person change itself into a normal Person using the Change Pattern. A bug in the Simulation Creation Toolkit, however, stops any agent from Absorbing itself. Therefore, users had to figure out that the Background Agent could be used to Absorb a Sick Person or change a Sick Person back into a normal Person.

Four participants created a perfect simulation; surprisingly 2 of them were not the participants who had prior exposure to Computational Thinking Patterns at the Summer Institute. Participant 1 had the last three interactions occur once every 0.5 seconds. This was probably an honest oversight and hard to debug as the simulation would look normal when run. Similarly, Participant 2 had the Sick Agent make a normal Agent sick once every 0 seconds. This mistake is easier to catch as it renders the 30% chance essentially useless as the rule gets executed at the fastest rate possible almost guaranteeing that the normal Agent will get sick. However,

Participant 2 saw an example of each interaction being executed and thought that the simulation was running as it should. Participant 5 had the Sick Person change the normal Person into a Sick Person; However, this Participant also selected the "For All Depictions" checkbox specification for both the Sick Person and the normal Person. This meant that a normal Person could change another normal Person into a Sick Person which was incorrect.

Finally, Participant 5 had the Background agent Absorb the Sick Person agent with a 10% chance; however, this occurred in any direction rather than in just 1 direction. Therefore, a Person Agent had the four Background Agents around trying to absorb it with a 10% chance each every second increasing the effective percentage. This was apparent when Participant 5 ran the simulation and all the Sick Person Agents died off immediately; however, it was not clear to the Participant why this was occurring. Furthermore, Participant 5 had no prior experience with AgentCubes and thus was confused by the concept of depictions. This is a good example of how the Simulation Creation Toolkit does not give users clues as to how different specifications interact with each other within a pattern.

5.4.3 Predator/Prey Program Results

The Predator/Prey simulation participants were asked to program involved the following ten interactions:

1) (normal) Fox moves randomly.
2) Rabbit moves randomly.
3) Fox gets hungry.
4) Hungry Fox (depiction of Fox) Tracks Rabbit.

5) Hungry Fox gets to Rabbit it kills it
6) Hungry Fox becomes a regular Fox after eating a Rabbit
7) Dead Rabbit eventually decomposes
8) Hungry Fox can die of hunger if it does not eat
9) Foxes reproduce with other Foxes
10) Rabbits reproduce with other Rabbits.

The following table displays the Analogical Reasoning Study results for the Predator/Prey program; the participants highlighted in yellow refer to people who were involved in the Summer Institute and the participants highlighted in green refer to people who were not.

	Duration	**Inter. 1**	**Inter. 2**	**Inter. 3**	**Inter. 4**	**Inter. 5**
Participant 1	40 min	√	√	√	√	√
Participant 2	30 min	√	√	√	√	*X*
Participant 4	32 min	√	√	*X*	√	*X*
Participant 3	25 min	√	√	√	√	√
Participant 5	1 hour	√	√	√	√	√
Participant 6	1 hour	√	√	√	√	*X*
		Inter. 6	**Inter. 7**	**Inter. 8**	**Inter. 9**	**Inter. 10**
Participant 1		√	√	√	√	√
Participant 2		*X*	√	√	√	√
Participant 4		*X*	√	√	*X*	√
Participant 3		√	√	√	√	√
Participant 5		*X*	*X*	√	*X*	*X*
Participant 6		*X*	√	√	√	√

Table 15: Predator/Prey Results For The Analogical Reasoning Study

Only two participants were able to implement this simulation perfectly correct—one from each group. The Predator Prey simulation is the most challenging of all the programs in the Analogical Reasoning study. Participants took significantly longer on this simulation than the other two programs. This is partly because it contains the most patterns and also because a few of the interactions, like the Epidemiology simulation before it,

take some creativity to execute using the Simulation Creation Toolkit. Like the Epidemiology unit, Foxes need some mechanism to make them hungry. Participants used the Change Pattern to execute this. Some participants had the background Dirt Agent change the Fox into a Hungry Fox. Other participants had the Fox change itself into a Hungry Fox. Either way, the interacticon animations do not readily lead participants to these interactions. Similarly, to have the Hungry Fox change back into a normal Fox takes a little ingenuity using the Simulation Creation Toolkit. Since the Change pattern cannot be implemented as an action of another pattern (i.e. change the Fox back into a normal Fox if a Fox kills a Rabbit), the only way to implement a Hungry Fox becoming normal is to have the Dead Rabbit Agent change the Hungry Fox Agent back into a normal Fox. This confused many participants because it was not readily apparent that this could not be completed as a specification to the pattern wherein the Hungry Fox changes the Rabbit into a Dead Rabbit.

Participants 2,4, and 6 were not able to figure out that the Dead Rabbit changes the Hungry Fox back into a regular Fox. Instead, these Participants had the regular Rabbit change the Hungry Fox back into a regular Fox and the Hungry Fox kill the regular Rabbit. This led to a situation wherein either the Hungry Fox would either become a regular Fox when it got to a Rabbit Agent, or the Hungry Fox would kill the Rabbit Agent but stay hungry; both could not happen. However, from these participants' point of view Foxes were becoming hungry, eating Rabbits and also changing back into normal Foxes, and, therefore, all the simulation interactions were present.

Participant 4 and Participant 5 confused the Hungry Fox with the regular Fox throughout the simulation. This led to the situation wherein the

Foxes all started out hungry and immediately killed all the Rabbit agents. The Simulation Creation Toolkit allows users to pick from Agent with their various depictions; however, it is easy for a user to overlook the name of the particular agent they are selecting. Since the pictures on the agent pop up menu are small, if two agents look alike and a user does not notice the label, the user can easily pick the wrong agent.

Participant 5 was not able to get the Hungry Fox to change back into a regular Fox and eventually gave up on creating the simulation. This is a major issue with the Simulation Creation Toolkit, namely, unintuitive mechanisms for creating certain interactions could lead to users being turned off by the interface as they do not have the control they need to create the interactions they want. Instead they must somehow make the interaction they want fit in with premade patterns.

Finally, including this simulation was in-part to better understand if middle school students would have been able to create their simulation without guidance. It is fairly clear that this would not have been the case given the unintuitive nature of the patterns. It is also clear that this simulation is probably not a good introduction to the Simulation Creation Toolkit as the patterns do not lend themselves as easily to the Predator/Prey interactions as they do to the Epidemiology simulation. The solutions to this issue might include enabling a pattern to be triggered in concert with another pattern execution or more specifications for patterns, like a flag that can get set when a pattern occurs. Thus, the user could use one of these mechanisms to figure out when to trigger the Hungry Fox changing back into a regular Fox for example. Even with these issues, two users were able to implement all the patterns correctly including a user who was not part of the Summer

Institute and two other users were just two patterns away from a perfect simulation.

5.4.4 Discussion Of The Analogical Reasoning Study

This section will review the Analogical Reasoning Study taking into consideration the notes obtained while users were going through the study, the responses users had to the feedback questions, and the simulations users themselves created. Specifically, this section will look at system ambiguities that participants encountered, instances of participants of using the Simulation Creation Toolkit for debugging logic errors in their programs, and system bugs. Appendix F displays the participant answers to the two feedback questions; excerpts from these will be used in this section.

The specification for the Absorb Pattern forces users to select the absorption to happen in a specific direction or if it is stacked on another agent (see Figure 66). This confused users because it entailed reading the specification closely to see that one or the other had to be selected. Furthermore, some users decided to click the checkbox stating that the pattern happened in any direction; however, without selecting the checkbox that specified the direction choice, this selection was rendered useless. A related issue is that the Absorb Pattern is similar to the Change Pattern but the Change Pattern does not allow for an agent to change another agent when stacked in some formation. Some participants became frustrated with this because they had to find other ways to change a given agent which was not the way they wanted.

Many participants pointed out that the interacticon for the change agent does not make it apparent that one agent could change itself. The interacticion makes it seem like two agents must be present in the change

interaction to work not making it apparent that if the center direction is specified it works on the agent itself. Participant 1 stated the following:

"I would add a feature for an agent to change its own behaviors…"

Participant 2 echoed these sentiments stating:

"add 'self' or a way to tell the user that an agent can trigger something on itself."

Furthermore, having the background change an agent when no specific interaction leads to that agent changing, was a leap that many participants thought was not afforded by the system. Participant 2 stated

"Having actions triggered by the floor and not by the agents doing the action is non-intuitive."

Having a Background Agent make the change also leads to other errors. For example, enabling the Background Agent to change an agent in any direction might change a specified percent chance that the change is executed by making it four times more likely. Participant 2 stated

"Also had trouble getting foxes not to die because I had them being polled in all directions, the likelihood skyrocketed."

Making it more clear that a certain pattern can be executed without interaction with another agent could solve these issues and this problem should be fixed for future iterations of the system.

The specification for the Tracking Pattern led to some confusion among participants. The Tracking Pattern only works if at least one background agent is specified for the tracking agent to move on (see section 3.6.6). However, the pattern does not force a user to select a tracking agent, and thus, the pattern could be created with no background agent meaning the pattern will not execute. Many participants had initial trouble with this, implementing the tracking pattern in the Pacman program and being

perplexed when the tracking agent did not move. Another issue is that the mechanism for choosing an agent for the Tracking Agent to move one looks almost identical to the mechanism in every other Movement Pattern wherein users pick what agents block the movement. At one point during the creation of Pacman Participant 1 exclaimed:

"Oh, is allowed to move ON!"

The Simulation Creation Toolkit should make it clear that this specification must be picked in order for the pattern to work.

The notes show multiple example of participants who had never used AgentSheets/AgentCubes before and never being exposed to patterns, not understanding how various interactions work. Participant 6 wondered allowed the following, for example:

"Which pattern allows an agent to pass through another agent?"

For example, what would allow a Ghost to pass through a pellet when creating Pacman? In the Simulation Creation Toolkit this is a pattern specification of the Tracking Pattern. In AgentCubes, the default move behavior is such that an agent can move on any other agent unless conditions are added such that it is blocked by a given agent. This is somewhat analogous to the Simulation Creation Toolkit where agents can move on other agents unless the blocked by specification is picked (like with Random Movement) or the allowed to move on specification enables this movement (as with the Tracking Pattern). Participant 3 and Participant 5 asked at one point during one of their program creations:

"What does stacked mean?"

The idea of one agent stacking on top of another agent is something intuitive for a user who has seen AgentSheets/AgentCubes before but not for participants who had never seen this. Participant 6 and Participant 3

initially thought that the Wall Agent should push the Pacman Agent because Pacman cannot go through walls. For someone who has never seen the Push Pattern before, this could seem logical as the Wall Agent is pushing back on Pacman's movement.

At one point Participant 5 asked the following:

"Does dirt have to move to make it disappear."

This question was asked as the user was trying to make the Dirt Agent absorb the Rabbit agent. This relates back to the idea that interacticons, though helpful when they closely mimic the interaction a user is currently creating, can also be misleading if the interaction looks significantly different. As will be talked about in Chapter 6 there are many ways to correct this ambiguity. One involves giving agents multiple interacticon examples for the different ways a pattern can be used. Participants without prior AgentSheets/AgentCubes experience had more trouble with the idea of depictions. Namely, when to specify a pattern for all depictions of a given agent and when to not. Participant 5, for example, implemented both the normal Person moving randomly and the Sick Person moving randomly as separate patterns in the Epidemiology Simulation. In subsequent versions of the system it might be necessary to explain not only the pattern but what each specification does more clearly to guide users through not only picking the correct pattern, but also, finding the correct specification.

Finally, both types of participants did not clearly understand what the direction specification meant in different contexts. Namely, why does the center direction specification refer to the agent itself in some scenarios but refer to the top of the stack in others. For example if an agent looks to its right it can only see the top agent to its right. However, if it uses the center specification it can only see itself even if it is not on the top of the stack in

that particular square. However, if an agent moves using the center specification, that agent is placed at the top of the stack. This ambiguity is not exclusive to the Simulation Creation Toolkit as it is shared with AgentCubes but it is something that throws off users. Participant 6 stated the following in relation to the direction specifications:

"The directional elements were unclear. (drawing of the directional specification) is only useful in some cases…"

There were many instances of the system enabling and hindering attempts at debugging. Most participants play tested their games as they created patterns. If they saw the interaction happen, they would assume the interaction was correct. This worked in many cases but in some cases this misled the participant into thinking that a pattern was right when it was not. The cases where it worked were instances where it was obvious the pattern was not occurring. In the Tracking Pattern, most users neglected to select a background agent the first time they implemented it. After running the simulation and noticing that the tracking agent was not following the tracked agent, they went back to the specification and corrected it. Incorrect implementations were not always blatantly obvious however. For example, if a percent chance specification is incorrect or a once every specification is incorrect the user might see the interaction occur but it would not be happening at the rate that it should. An example of this was Participant 1's Epidemiology simulation wherein patterns occurred once every half second as opposed to once every second. Similarly, in the Predator/Prey simulation, if a participant had the Hungry Fox kill the Rabbit and the normal Rabbit change the Hungry Fox back into a regular Fox, both interactions would be present, just not occur together. Therefore a user would see both of the

correct interactions occurring and might incorrectly think that the simulation implementation was correct.

Some participants used the patterns and specifications to enable debugging. Participant 1 and 2 often changed the percent chance specifications for patterns to 100% to ensure the pattern was executing before moving onto the next pattern. Participant 2 and 3 decided to implement the Data Count Pattern to see if any Sick People agents were dying in the Epidemiology simulation.

Some users found the interface problematic for debugging. Participant 2 stated that being able to manually order the patterns, though it would not change the simulation, would be helpful to the user. Participant 4 echoed these concerns. Both of these users were Summer Institute attendees and thus were used to moving around rules. AgentCubes necessitates rule movement because only one rule of every method is executed. However, it never occurred to me that there would be another reason that users might want to change the order of rules. This functionality should be included in subsequent versions of the system. Participant 6, who was not a prior AgentCubes user, stated that it was easier to delete all the patterns and start over rather than debugging. This sentiment was not shared among the other participants, however, it illuminates the fact that the user interface is still at a level that novice users might find confusing and must be improved in terms of debugging in subsequent iterations of the Simulation Creation Toolkit.

Some system bugs were discovered as participants created their programs. Most have been talked about in previous sections. Things like no stacked specification on the change pattern and not being able to specify the tracking agent as an agent to be moved on should be corrected in the tool. One user pointed out after deleting a pattern that there should be an undo

button. This is not a bug per se but a feature that should definitely be added to future iterations.

Only one participant was able to create all the simulations perfectly, Participant 4. Surprisingly, this participant was one who had no prior experience with AgentSheets/AgentCubes. This participant actually appreciated many of the above ambiguity issues with the system. For example, in the feedback questions this user stated:

"By limiting the number of interactions available to the agents, the tool actually promoted thinking about how to make agents do what the program needs…"

On the second question Participant 3 went onto state the following when asked what pattern(s) she/he had trouble with:

"Wall changes person to sick person. I realize that having the sick person act on itself might have beeneasier, but I'm glad that the tool allowed for roundabout methods of problem solving, counter-intuitive as they may be."

Though this reaction to the system might be atypical, the fact that a person with no AgentCubes/AgentSheets experience was able to create all these simulations gives future hope that participants with little to no prior experience with Computational Thinking Patterns or the mechanics of AgentCubes could use the system successfully. This might be especially true if the above criticisms regarding system ambiguity and system bugs are met as well as more methods that enable users to debug their simulation.

CHAPTER 6

6. DISCUSSION

This chapter will discuss the results and how they relate to the research questions. It will go into what can and what cannot be said about the research questions based on the data acquired during the course of this thesis. As a point of review, the two research questions that are being analyzed are the following.

RQ1: Can students programming at the Computational Thinking Pattern level successfully create simulations of scientific phenomena they are presented within class?

RQ2: Does students programming science simulations using high level Computational Thinking Pattern lead to better student conceptualization and understanding of the material they are being taught?

We will look at each research question in terms of the different results presented in the previous chapter; these include the simulation creation exercise, the worksheet questions, and finally the analogical reasoning study. Finally there will be a general discussion section that talks about anything that was not covered in the results discussion.

6.1 Relationship Between Study Results And Research Question 1

The first research question focuses on the ability of students to use the tool to implement simulations. This question focuses on whether students

can use the Simulation Creation Toolkit to create simulations of phenomena they are presented with in-class. In this case students used the Simulation Creation Toolkit to create a Predator/Prey simulation. Since the study was accomplished in a highly guided environment, this in-class portion of this study looked partly at how quickly students could create simulations with guidance and how well students were able to work the mechanics of the system itself. Furthermore, various worksheet questions give insight into how students might use the system in subsequent exercises and what students understand about what they have created thus far. Finally, a study wherein older students, with very little guidance, create simulations from high level descriptions will be analyzed to discover if users can actually build simulations from analogy with little direction.

6.1.1 How Results Relate To Research Questions

As mentioned before, students created the Predator/Prey simulation in a highly guided environment. The students were provided with, for the most part, step by step instructions on how to create the simulation. The simulation creation portion of this study aims to look at whether students can create a simulation using the system using the guidelines they are given. To put another way, could students successfully work the mechanics necessary using the Simulation Creation Toolkit to successfully create a simulation.

The results for the seventh grade Life Science students showed that students could indeed follow the directions necessary to create a simulation. On the partial credit scoring of the patterns, the seventh grade class as a whole on average obtained at least an 85% or above correct on each pattern. The all or nothing scores reveal that at least 80% or more seventh grade students implemented each pattern correctly. Furthermore, 40% of seventh

grade students had a perfect simulation implementing all 16 patterns correctly and 80% of students had only two or less mistakes on all 16 patterns.

The results for the sixth grade class also showed that the system, in a heavily guided environment, also is effective with students who had never seen AgentSheets before and created the simulation out of context. As mentioned above the sixth grade students only had 3 days to complete the simulation which ends with pattern 10 being implemented. Through the first 10 patterns, the ones covered in the first 3 days, the sixth grade students virtually matched the seventh grade students in terms of partial credit and all or nothing credit on pattern implementation (see Figure 90 and Figure 92). Through the first 10 patterns, the lowest all or nothing credit pattern score was 86% and the lowest partial credit pattern score was 90%. Moreover 23% of sixth grade students had all 16 patterns correct and 47.6% of sixth grade students had a perfect 1-10 patterns implemented. 80% of sixth grade students had 1 or less pattern wrong in the first 10 patterns.

This data tends to show that students were able to successfully create simulations given heavy guidance using the system. Furthermore, many students would implement the correct pattern but miss a specification relating to the pattern. This is further shown when looking at the drop-off between the partial credit score and the all or nothing credit score. The specification for a pattern could be hard to implement for a variety of reasons. For one, the pattern itself shows large versions of the agents acting out the interacticons. If a student picked a wrong shape of a given agent for a pattern, it would become obvious very quickly. However, if a student picked a wrong agent as a blocking agent on the Random Movement Pattern, for example, the student must either astutely recognize the text below the menu,

that displays the name of the agent and the name of its shape, or be able to see that the small picture of the agent they have selected is not the correct shape of that agent. This could be confusing, for example, when trying to tell the difference between a Hungry Fox Agent and a normal Fox Agent because the normal Fox Agent is brown and the Hungry Fox Agent is red, however, with such a small picture they are hard to tell apart. Future iterations of the system should make these agent shape pictures bigger to enable users to better understand which shape they are choosing.

Choosing the wrong specification does not necessarily mean the simulation will run incorrectly or be decipherable except in specific situations. This further hinders implementing a pattern completely correct as students not only have to notice the subtleties involved in picking the right agent shape, but if they pick the wrong one, they might not readily see it when they run the simulation. For example, the normal Fox Agent randomly moves and is blocked by the depictions of all other animal agents (among other blocking agents like the Wall). If a student overlooked picking the Dead Fox agent as a blocking agent for the normal Fox Agent Random Movement Pattern, the student would only be able to notice this if a regular Fox Agent happened to jump on top of a Dead Fox Agent during the simulation run. Given that the Dead Fox Agent decomposes into a Grass Agent in 4 seconds, and that the simulation has approximately 100 agents moving around at the same time, a student might never see that the regular Fox Agent jumps on the Dead Fox agent; this would involve a student looking at the right Fox Agent at the exact right time. Therefore, the student might never correct this and it would show up as an incorrectly implemented pattern.

Furthermore, there is an argument to be made that missing this specification does not actually change the simulation substantially. If a

regular Fox Agent, for example, is able to walk on a Dead Fox Agent, the regular Fox Agent will still eat, breed, get hungry, die etc. This specification is useful in some situations, such as if a Fox Agent is stacked upon a Rabbit Agent, that Rabbit Agent's scent will not be diffused and a nearby Hungry Fox might not track it. Furthermore, aesthetically, the simulation is a lot easier to decipher if agents are not obscuring other agents via stacking. However, the ability of a Fox Agent to walk upon a Dead Fox Agent does not change the simulation functionally. The student would still be able to run experiments on the simulation and obtain sensible results.

This speaks to a larger issue in this study. When using the Simulation Creation Toolkit, students can conceivably create many different Predator/Prey simulation implementations all of which are valid. These different simulation implementations could have aspects to them that are more realistic or less realistic when compared to one another. The instructor must decide what aspects of the simulation might work for the purposes of the class and what aspects might be too advanced or unnecessary to have students implement. Therefore, the ability of students to follow the lead of the instructor when using the Simulation Creation Toolkit, whether that be in walkthrough form akin to the study done in this unit or a higher level description of the simulation (or maybe some combination thereof), is crucial. This study shows initial hope that students could complete a guided unit using the Simulation Creation Toolkit. It would be interesting to study to what extent and the different ways teachers could use this tool in their classroom and what strategies work the best and what needs to change about the tool to better accommodate teachers, including units that are less guided.

An ideal study would have had students create simulations over the course of the semester. The first unit, under this scenario, could be highly

scaffolded. After students are introduced to the system through this initial unit, subsequent units could be less guided and possibly more open-ended. In this scenario, a unit akin to this study, wherein students are given a walkthrough to complete the simulation would be the first step. In fact, before doing a larger scale study it is essential to complete a study where students are given guidance for if the results were to show that students cannot accomplish this task, then a larger study should not be tried until changes are made that enables students to better use the system or a better strategy on how to integrate simulations into classrooms is developed. As the results show, many students went from being introduced to the tool to creating a full simulation in 4 days or less. As a proof of concept, the Simulation Creation Toolkit and the strategy of using Computational Thinking Patterns to create simulations looks like a promising avenue for integrating computational thinking concepts into the classroom.

There are a few pragmatic reasons for scaffolding the unit. First off, the student population that participated in this unit at Centenary Middle School involved a variety of students of varying backgrounds including English as second language speakers and learning disabled students. Giving students step by step guidance through pictures enabled more students to have the ability to finish the simulation. Secondly, this unit takes up a significant amount of class time. To hit all the GVC items previously outlined, necessitates that students get as far through the simulation creation exercise as possible. Given that this is the first large scale study of an experimental system, the walkthroughs allowed students to systematically create the simulation being exposed to these GVC concepts along the way.

Regardless of the reasons, the above results should be taken for what they are. The results indicate that students, when given guidance, could use the simulation creation toolkit to create the Predator/Prey simulation. Future research should focus on methods of removing this scaffolding to see how well this system might work when students are forced to program scientific phenomena they are currently studying purely by analogy.

6.1.2 Discussion Of How Selected Worksheet Question Results Relate To Research Question 1

Some of the worksheet questions give insight into how well students might create subsequent simulations without the guidance provided in the Predator/Prey unit. Questions 4, 7, 10, and 13 asks students to identify the pattern they would use to implement a given interaction. The results for these worksheet questions, presented in section 5.3.2, are mixed. Disregarding possible question ambiguities discussed in section 5.3.2, seventh grade students got these questions 70% correct. Sixth grade students, on the other hand, got these questions correct 50% of the time.

This is one of the few areas where sixth grade students scored lower than seventh grade students. There are a few reasons that this might be the case. First off, seventh grade students had been exposed to Predator/Prey interactions before. Therefore, seventh grade students could rely on prior knowledge of what the simulation should look like to reason out what subsequent interactions need implementing. For example, when the Hungry Fox Agent chases the Rabbit Agent these students might be more apt to realize that the Hungry Fox Agent should eat the Rabbit Agent at this point and become not hungry. Sixth grade students, however, have only the intro exercise to draw upon wherein each student plays a specific role in the simulation (see section 4.3.1). This intro exercise took about 10 minutes and

was the first exposure many of these students had to the Predator/Prey simulation. Therefore, these students could have easily overlooked the fact that the Hungry Fox Agent should eat the Rabbit Agent and instead thought that the animals still needed to mate, for example, yielding an incorrect answer. It should be noted that many students who got this question wrong stated the wrong interaction that needed to be added but actually specified the right pattern for this question (they would say the Hungry Fox Agent still needs to mate and that the Generation Pattern would be used for this—this answer is technically right but not in the context of the question).

As mentioned before, simulations of the same interaction can be constructed with differing choices as to the interactions to include and not include. When using the Simulation Creation Toolkit to model interactions for a specific simulation, a priori knowledge of what interactions are important for a particular unit and what interactions might not be important could help students know what patterns still need to be implemented. Since seventh grade students worked on the Predator/Prey simulation as part of a larger study into ecosystems, and the simulation was developed working with their teacher, the students may have had a much better idea of what interactions and patterns to include as they created the simulation. The sixth grade students, on the other hand, had just been introduced to the concept of the Predator/Prey simulation; though one might surmise that after tracking the Hungry Fox Agent should eat its prey or the Hungry Rabbit Agent should eat the grass, for example, it is possible that a student might overlook this if they had not been exposed to the Predator/Prey simulation before. If a sixth grade student assumed that the Hungry Fox Agent tracking the Rabbit Agent ended the eating interaction it makes sense that the student might guess that the Hungry Fox Agent still needs to reproduce for example. It is possible

that seeing a few video clips of Predator/Prey interactions might have helped the sixth grade students better understand all the interactions in the simulation and possibly stopped them from overlooking a pattern.

These results give initial hope that students can use the Simulation Creation Toolkit to pick the correct patterns in subsequent simulations. Namely, a majority of students were able to pick the correct pattern to implement a given interaction. Seventh grade students were able perform better at this task than sixth grade students but both groups seemed to be able to select the correct pattern when necessary. In terms of Research Question 1, however, one would want students to overwhelmingly pick the correct pattern in each situation if they are expected to pick the correct pattern after the guidance is removed, especially on the final questions. In this respect, though this initial data is promising, more work has to be done in both the presentation of the system itself and introducing the system to students such that they are able to better pick the patterns necessary to properly create a given interaction. Given that this is students' first experience with this tool and the idea of programming at the pattern level, it is realistic that students might perform better in subsequent simulations.

Questions 14-21, and 25 focuses on students running the beginning of a simulation using the Simulation Creation Toolkit. This part of the unit gives insight into students' abilities to use the system to create a hypothesis, change pattern specifications to fit this hypothesis, run a single trial of an experiment using this modified pattern specification, checking to see if the result was reasonable, and coming up with a reason why this result was obtained. The result for Question 17 show that most students had a sensible hypothesis and the results for Question 18 and 19 show that most students had sensible results meaning that even if their simulation was created

incorrectly, it was correct enough to yield reasonable results given the parameter change. This data seems to indicate that students are able to create simulations and run experiments on this simulation.

Question 25 asks students to create a Poacher Agent and a Bullet Agent with very little guidance using a pattern they had not yet used—the Directional Movement Pattern. Furthermore, there were many correct ways to implement this interaction—students could have the Bullet Agent change the Fox Agent into a Dead Fox or absorb the Fox Agent. 27 seventh grade students got to and completed the Poacher Agent. The results for Question 27 show that most students had a reasonable answer as to how the Poacher Agent effected the population. Moreover, running the simulations that students created showed that the Poacher Agent was implemented correctly in almost all of these simulations. As with the above data on worksheet questions, this seems to indicate that students can use the Simulation Creation Toolkit even when not directly guided with step by step instructions as to what to do next.

In terms of Research Question 1, the above data seems to show that students have the ability to identify what future pattern to use, complete the simulation in a way that is correct enough to get sensible simulations results, and create agents from scratch using the Simulation Creation Toolkit to add interactions with little guidance. Finally, students were able to get to this point after 4 days or less of use; if used over the course of the semester students might be able to use the system more effectively and with more freedom to create simulations. Further research must still be done, however, as a proof of concept, these results look very promising.

6.1.3 Discussion Of Analogical Reasoning Study Relates To Research Question 1

Research Question 1 asks if students can create simulations in the classroom using this tool. The Simulation Creation Toolkit affords two ways to facilitate simulation creation. One advantage of the Simulation Creation Toolkit is that it takes users less time to implement certain patterns. A good example of this is the Tracking Pattern implementation (see section 1.4.4). Another advantage of the Simulation Creation Toolkit is that it potentially allows users to create simulations by analogy. However, given the constraints of the classroom environment, this functionality was not tested in the study integrated into the classroom. In order to see if this functionality actually exists in any meaningful way, an the Analogical Reasoning study was added to see what this tool possibly affords users in terms of creating simulations purely by making an analogy using the interacticons and the descriptions they see.

The Analogical Reasoning study had too small of a sample size to generalize the results to all users. Furthermore, the study involved a different environment than the classroom as well as a population of participants that were vastly different than middle school students. However, the results of this study do give initial hope that students could possibly create simulations by analogy especially if corrections to the system are made.

The first two programs created using the Simulation Creation Toolkit, Pacman and Epidemiology, were almost created perfectly by all users with slight errors. Even the unintuitive interactions of the Epidemiology unit were eventually debugged and made mostly correct. Both users with prior AgentSheets/AgentCubes exposure and users with no prior exposure did well

on these programs. The only big difference between the two populations was the amount of time spent creating the simulations with novice users taking more time in general.

The simulation that caused users most trouble was the Predator/Prey simulation similar to the one taught at the middle school classroom. This points mostly to the fact that this simulation has interactions that are not completely intuitive to users as the Simulation Creation Toolkit is currently configured. Specifically, the programming by analogy advantage of the system breaks down if the analogy itself is faulty of misleading. Given that many of these participants were not able to create some of the interactions correctly points to the fact that middle school students probably would have had trouble creating the Predator/Prey simulation via analogy. Adding functionality that enables patterns to be executed in concert as well as a more comprehensive interacticon system could help make creating this program as easy as creating the Pacman game and Epidemiology simulation.

This study showed that the Simulation Creation Toolkit can enable programming by analogy but only when the analogy fits the mechanisms provided by the Toolkit. As pointed out in section 3.2.1, many simulations could potentially be made using this program. However more research must be done into whether these simulations are ones that users can realistically implement even if the tool allows users to implement each interaction. The central question of this further research might be can users employ the system to create the following interactions by analogy. Interaction specific testing using the system could yield a better idea as to what is realistically possible as opposed to what is theoretically possible using the system.

6.1.4 Discussion Of How The Worksheet Results Relate To Research Question 2

Research Question 2 looks at if creating simulations actually helps students understand in-class material any better. The idea behind this question was that students would create a simulation and by going through the creation steps in addition to running the simulation, the students would better understand related in-class concepts related to the simulation they were creating. The results of this question were inconclusive.

There are a few reasons that this study was not able to answer Research Question 2. The first is that the actual act of creating simulations showed little or no correlation to the ability to answer worksheet questions correctly. As discussed in-depth in section 5.3.1, there are many possible reasons for this including the idea that concepts from Guaranteed Viable Curriculum do not lend themselves easily to simulation creation and the created simulations themselves might not be realistic. Section 5.3.1 also points out that though the simulation creation exercise itself may not directly lead to better understanding, the simulation creation exercise did give students an opportunity to be exposed to or revisit oft-misunderstood concepts from the GVC.

However, the experiment section, Questions 14-21, and 25 hit multiple parts of the GVC directly. These sections include LS15-A which is how technology enables easier experimentation, LS3-B and LS3-C where students make predictions or hypothesis, and LS12-A where students change parameter values in pattern specifications to try to alter the Fox Agent or Rabbit Agent population in some way. Furthermore, the results for this section seem to indicate that seventh grade students could complete all the

concepts associated with the Guaranteed Viable Curriculum (as noted above not many sixth grade students got to this point).

This part of the GVC however, relate directly to simulation creation—namely creating models, predicting, taking data, analyzing data etc. This is not typical of most of the GVC concepts specifically associated with ecosystems for example. These concepts involve ideas such as knowing what decomposition is, understanding concepts such as trophic pyramids or the fact that humans directly and indirectly effect populations. These are all concepts that the simulation touches upon but are not at a level that the simulation creation activity might help students better understand the issue because they are at such a basic level.

Research Question 2 might have been an ill-posed question within the realm of this study. This study is an initial exploration into using patterns to create simulations. The idea that students might actually gain greater insight into topics covered in the GVC, consisting of curriculum requirements that have previously been taught successfully without such activities, could have been too ambitious. Given that this system had just been created, it might have been better to focus the Research Questions in the area of the ability of students to create simulations. Then subsequent research could have focused on greater understanding of material post simulation creation.

Another problem with Research Question 2 and this study, briefly talked about before in section 5.3.1, is that to answer the question affirmatively involve proving that students gained knowledge they previously did not have during the simulation creation activity. The easiest way to do this would have been to give students a pre and post test and seeing if the post test scores were significantly better than the pretest scores on questions

related to the GVC. A second method for figuring out whether the simulation creation activity actually enhanced student understanding was suggested by Dr. Clayton Lewis who mentioned that if a group of questions could be identified wherein the student could not answer it without creating the simulation correctly up to a given point, then that question could indicate whether the simulation creation activity actually helped students better answer this group of worksheet questions. Unfortunately, as explained in section 4.2.3, none of the questions asked fit this criteria. Given that these questions were taken from student misconceptions of ecology in the GVC, many of these questions might have been helped by creating the simulation and running the simulation but students could readily get the answer by recalling the idea from a previous class lecture.

As mentioned above, the one area where this activity seemed to directly relate to the GVC was when students ran part of an experiment on the simulation they created and added a Poacher Agent. However, it is impossible to tell if students misunderstood these GVC concepts before they ran the experiment and if they understood these concepts any better after they ran the experiment. All we know is that these topics were important according to the teacher and that students often had trouble comprehending these concepts, and thus, it would be helpful to have this unit touch upon these ideas.

As noted in section 5.3.1 (and will be reviewed here briefly), even though the simulation creation activity did not correlate with students answering the questions any better, the activity did expose students to concepts that were often misunderstood in the GVC. The way the GVC is currently written, does not lend itself easily to modeling activities; therefore, integrating a simulation creation unit with concepts that touch upon various

GVC elements might be an effective strategy for integrating computational thinking concepts into the curriculum. Basically, using this strategy teachers do not have to choose between doing a modeling unit and covering the GVC material through other activities.

6.2 General Discussion Of The Project

There are many points to discuss when talking about an initial proof of concept study pertaining to a brand new system. The unexplored area of creating simulations through employment of Computational Thinking Patterns yields many items that should be noted in this study and/or changed in subsequent studies. Furthermore, a discussion of what would need to happen to have teachers realistically use such a system in class is an essential step in reaching the overarching goals of the Simulation Creation Toolkit.

Mentioned before but bears additional discussion is shortcomings of the study design. Ideally, to answer Research Question 2, students would have been provided with a pre and post test to truly quantify if the act of creating a simulation helped enhance student understanding of various GVC topics. Furthermore, though the heavily scaffolded environment wherein students created the simulation was deemed essential taking into account the realities of doing the study in-class, it brings into question the extent students actually used the system as intended as opposed to just following instructions. Questions that asked students what pattern they would use to implement certain interactions, the analogical reasoning study, and the addition of a Poacher Agent with little direction all point to the fact that this system can indeed be useful once the scaffolding is removed; however, it

might have been better to do the study in an environment wherein the heavy guidance was not necessary or could be removed.

Somewhat related is the idea that the ambitious nature of the Research Questions did not match the limited time and environmental constraints of the study. To answer Research Question 1 fully we would need to see how students created a scientific phenomena using the system with little guidance; otherwise Research Question 1 is only answered affirmatively in the sense that if students are given step by step instructions, they can create simulations. Research Question 2 could only be answered affirmatively given that the simulation creation activity actually helped students understand GVC items better after the activity was completed. Given this study, though the simulation allowed students to touch upon many GVC items, it is hard to say whether students understood any of them better because of the simulation programming activity.

A larger scale study would have a much better chance of answering these Research Questions more fully. Given more time and multiple units to try this activity on, students could have used the first unit to get acquainted with the idea of creating simulations using the Simulation Creation Toolkit. Then subsequent units might have had less guidance enabling students to create simulations purely by analogy. Doing a range of simulations would have provided insight into what types of simulations worked well with the tool and what types of simulations did not. Furthermore, a pre and post test could have been performed on every unit to see what types of GVC items were enhanced by the student creation of simulations and what types were not. However, if this study showed it was not possible for students to use this tool to create simulations in a heavily scaffolded setting, subsequent studies would not be necessary without a change to the system or its presentation.

Thus, this study could be thought of as a feasibility study or an exploration into this much larger study.

Another problem with the study had to do with the actual execution of the worksheet. In retrospect, many of the questions were ambiguous leading to students being confused or answering a completely different question in some cases. For example, when asked what the Hungry Fox still needed to do in the simulation, students responded with the idea of reproduction rather than eating the Rabbit Agent. Being that this is the first large scale study that I have done in a classroom, I did not realize possible ambiguities that might occur in these questions. Furthermore, running the questions by peers in a University setting is not the same as running the questions by 6th graders actually creating the simulation in a middle school classroom environment. More testing should have been done on the questions to better present the ideas of the worksheet more clearly. I also made the mistake of telling students that they could write "I don't know" if they truly did not know the answer to a question. I thought this would make answers more honest and better reflect what students actually learned from doing this activity and what parts of the activity had no effect on understanding. I also did not want students obtaining answers from other students. What I did not anticipate was that students might view the questions as a road block to creating the simulation and write "I don't know" for multiple questions in a row without actually attempting them. Given that this exercise had no bearing on their grade, there was no penalty involved for students doing this. Adding a pre and post test in addition to the worksheet questions might have led to clearer results as the test would be decoupled from the simulaton creation activity and given in a test environment, students might be more apt

to attempt reasoning through the question rather than writing "I don't know" on everything and moving on.

There are many things that should be modified in the Simulation Creation Toolkit. A few main changes that should be made will now be reviewed. The idea behind interacticons was that it would provide students with the ability to match the interaction of a particular phenomena they were seeing with the correct pattern needed to implement that phenomena in the Simulation Creation Toolkit. However, in some cases, the interacticon animation was misleading. For example, in the Change Pattern, a user might want an agent to change itself into something else. For example, instead of having the Dirt Agent change the Fox Agent into a Hungry Fox, it is just as valid to have the normal Fox Agent change itself into a Hungry Fox. However, the interacticon for this agent depicts one shape changing another shape into a third shape (see Appendix B). This could (and often did) lead users into thinking that the above interaction was not the Change Pattern or that the it was not possible using this tool. The interacticon depicts one implementation of the pattern—it assumes agent directions and movements--but there are many other possible implementations that are enabled by the combination of the patterns and their respective specifications. For example, the Generate Pattern, Absorb Pattern, Change Pattern, and Directional Movement Pattern show the pattern implementation happening to the right. If an agent is being generated to the left, however, when the agents are put in place of the generic agents the interacticon depiction looks wrong.

Another related issue is that inherent in the limited patterns palette, along with limited specifications for each pattern, some interactions have to be implemented via non-intuitive means. For example, in the AgentSheets Predator/Prey simulation agents get hungry after a certain amount of steps.

This is implemented using the Simulation Creation Toolkit by having the Dirt Agent change the Fox Agent into a Hungry Fox Agent. It is hard to make an analogy between the Dirt Agent changing the Hungry Fox Agent into a normal Fox Agent and a Fox actually becoming hungry out in the wild. As alluded to in the worksheet questions, the Fox Agent is using up energy as it moves and thus becoming hungry, but seeing a Dirt Agent change a Fox Agent into a Hungry Fox Agent is a much weaker visual analogy than the Fox Agent chasing the Rabbit Agent which students can and have readily seen in videos.

There are a few ways to solve the above problems. Future iterations of the Simulation Creation Toolkit could change an interacticon animation based on the specifications a user chooses. For example, if an Agent A already has random movement implemented for it and a user is trying to implement Agent A absorbing Agent B, then the Absorb Pattern interacticon can show Agent A randomly moving, and, if it bumps into Agent B, it absorbs it. This could be implemented by using an actual AgentCubes mini world with the two Agents rather than pre-canned animations. In this case, the system could enable just the behaviors that lead the agent to move and the current pattern. Another solution is to use multiple canned animations to depict all the different interactions that could be represented by that particular pattern and its specifications. The advantage of the first method is that it would add an element of instant debugging to the simulation. The advantage of the second way is that it would not be dependent on the already implemented code. For example, if a student has not implemented the movement of an agent yet, the first method would still not depict the correct interpretation. Furthermore, using the second method, depending on what sub pattern users pick, additional specifications could be filled in automatically. For example, if

a user selects a sub pattern wherein the animation has an agent being generated to the left, the specification for the Generate Pattern implemented could default to the left direction. Under both methods the interacticon animation could change as users made specification choices which could alleviate the confusion caused by displaying just one implementation of the pattern.

Debugging code using the system is not as easy as it should be. It is true that some participants in the analogical reasoning study were able to debug code using the specifications and or adding a data count pattern. However, most students at Centenary and participants in the Analogical Reasoning Study would not see bugs unless they were obvious in their simulation. A method that better clues the user into what was implemented might need to be developed. AgentSheets/AgentCubes dealt with this problem by introducing Conversational/Programming [61]; it is possible the Simulation Creation Toolkit could deal with this problem by showing the exact interaction between the two agents present in the pattern in the interacticon. The specifications in the pattern picker window do not visually differentiate patterns for students. Thus, students have trouble finding the exact pattern they implemented or sometimes, end up implementing two patterns that do the same exact thing. This can lead to weird behavior such as if an agent is randomly moving, the agent might randomly move twice each cycle instead of once skipping over two spaces. This should be changed by making each pattern implemented a different color.

Somewhat related, after this study it is clear that some patterns need to be modified. For example, the Generate Pattern should allow for an agent to check two directions before Generating a third agent (if Agent A sees Agent B to its left and Agent C in the downward direction generate Agent D).

A problem with the Predator Prey simulation was that Fox and Rabbit Agents could create new Fox and Rabbit Agents on top of other Fox and Rabbit Agents. If a population is mating out of control, this would actually lead to a positive feedback situation wherein Rabbit and Fox Agents are all conglomerated in the same area yielding unconstrained mating. Allowing the Fox Agent to check and make sure there is no other Fox Agent in the direction of the Generation Pattern would help temper this unconstrained mating. This becomes a problem because at some point, with enough agents in the world, the simulation will become slow and actually crash the program.

Somewhat related to both Research Questions is whether teachers will actually be able to use this system in the classroom. Since this was not explicitly studied it is hard to answer this question but a few things became apparent during the course of the in-class units. Marks Savs had made and taught simulations in AgentSheets before and therefore when problems occurred he was more apt to have a solution. For example, many pattern specifications relate directly to AgentSheets code, conditions and actions. Specifications like "this pattern happens once every 2.0 seconds" is something that a teacher with prior AgentSheets experience might realize that this is a once-every condition. Furthermore, implicit in this is that the rule is not guaranteed to fire once every 2.0 seconds, the time could be more depending on the simulation. This is a quirk of how AgentSheets/AgentCubes does timing but knowing these quirks sometimes help when trying to debug a simulation.

Moreover, having prior exposure to patterns also helps. Marks Savs attended multiple Scalable Game Design Summer Institute's at the University Of Colorado Boulder. Therefore, though Mark had never used the system before, he could figure out which patterns would be used for a given

interaction easily. Just from informal observation, this helped Mark be able to help students debug code especially if they missed a pattern or wrongly implemented a pattern. This knowledge of the patterns in combination with the pattern specifications I believe would help Mark be able to teach this unit using this system even if I was not present in the classroom.

Marco on the other hand had little experience with patterns or programming simulations. Marco could still help students with certain errors such as students implementing the wrong pattern, but for things like students picking a wrong agent depiction for a given pattern, it became a little harder to debug. This is partly due to differences in agent depictions being hard to see; however, with prior AgentCubes/AgentSheets experience one could run the simulation, see what agent or depiction did not have the correct behaviors, and then direct the student to modify the corresponding pattern in order to achieve the correct simulation behavior.

Both of these informal observations point to the fact that teachers would have the overhead of using the system for a few weeks before employing the unit in their classroom. Knowing the patterns and becoming comfortable with their implementations is essential to help students debug code that can become complex very quickly. For example the tracking pattern has 10 rules with over at least 3 agents if not more, each with a staggering amount of conditions and actions. It is not necessary for teachers to know the rules at the behavior level but it is necessary for teachers to navigate the patterns and specifications in the Simulation Creation Toolkit at a comfortable level.

An aim of this system was to enable teachers the ability to add simulation creation activities in the classroom without the deep overhead preparation, which normally goes along with simulation creation activities.

However, there is a tradeoff to using this system, namely, the overhead is shifted such that teachers must now get acquainted with the system. Once teachers know the system, they can enact simulation creation units in their classroom with much less overhead (time and preparation) than it would take to do the same simulation using lower level programming rules like if/then conditionality statements in AgentSheets/AgentCubes. It is not clear if this is something teachers would readily want to do as a way to introduce simulations into the classroom; more studies must be done on teacher perception of this tool to better understand if and how this tool might be used in the classroom environment.

CHAPTER 7

7. SUMMARY AND FINAL THOUGHTS

This thesis serves as a proof of concept for the strategy of using Computational Thinking Patterns to create simulations. Given the importance of integrating computational thinking concepts into the classroom, this initial exploration is a valuable data point from which to continue further investigations into this and other strategies.

This thesis presented a novel tool called the Simulation Creation Toolkit. This tool, built upon AgentCubes, enabled users to create simulations at the Computational Thinking Pattern level. The Simulation Creation Toolkit allows users to create simulations using combinations of 10 provided patterns with multiple pattern specifications corresponding to how the pattern is traditionally used. The tool implemented all the low-level AgentCubes behaviors for each agent present in the pattern automatically. In order for all the patterns to be guaranteed execution, a strategy involving timer methods was developed circumventing the AgentCubes restriction that only one rule in each method can be executed each cycle. The patterns themselves were modified and implemented such that they could be used in concert with one another including the implementation of modulating patterns, that change earlier or subsequently implemented patterns. The interface of the system allows users to choose from a palette of animated representations of all the Computational Thinking Patterns called interacticons. Once users picks a pattern, a window presents the user with all the implemented patterns in a list and the user can modify or delete patterns with the corresponding AgentCubes code being automatically updated as these changes take place.

Two studies looked at how effective the Simulation Creation Toolkit was in the classroom environment and specifically, tried to obtain insight into two related research questions. The first research question asked if students could use the system to create simulations and the second research question asked if creating simulations using the system actually increased student understanding of the curriculum material pertaining to the simulation. Between the in-class study and the analogical reasoning study it was found that students could use the mechanics of the system to create a simulation in a heavily guided environment regardless of prior simulation creation experience, and users could create simulations from a high-level description using the Simulation Creation Toolkit. Some worksheet results indicated that students could pick the correct pattern once the guidance was removed. Furthermore, students participating in the in-class unit showed that they could create a simulation in 4 days as opposed to a week when compared to high school students creating a similar simulation using conventional AgentSheets behaviors.

For a variety of reasons, no conclusion could be drawn as to whether creating simulations in the classroom actually help student understanding of the curriculum material. Some of these reasons relate to the level of knowledge students need for concepts in the GVC are not necessarily helped by running a simulation and the questions themselves did not require students to actually have a correctly created simulation and general difficulties associated testing student understanding. However, the unit did show that the simulation creation activity could work in concert with the GVC touching upon often misunderstood concepts as students created their simulations and students showed that they could use the system to make a

prediction and run part of a sample experiment which is a direct part of the GVC.

This study shows that the strategy of creating simulations at the Computational Thinking Pattern level looks promising. A larger scale wherein students do multiple simulations over the course of a semester could give greater insight into how well the system works for a variety of simulations and where the system is conducive to representing real world phenomena and where the system should be modified to better enable simulation creation. Before this larger scale study can occur, changes to the Simulation Creation Toolkit's interacticon palette as well as updated specifications for each pattern should be added. Finally, questions that not only target the GVC, but also, better tie into the actual simulation creation activity for each unit should be developed to better understand if creating simulations using this tool does in fact lead to better student understanding of the material.

By enabling students to create simulations, the Simulation Creation Toolkit can potentially facilitate computational thinking in the classroom environment. The initial data for this tool is encouraging. More studies have to be done to see to what extent this tool or similar tools can be fully integrated into the classroom curriculum. The initial results from this study show that programming at the Computational Thinking Pattern level is a promising avenue of further investigation.

REFERENCES

[1] Cooper, S., Dann, W., Pausch, R., Teaching Objects-first In Introductory Computer Science, In Proc. SIGCSE 2003, Reno, Nevada, USA, 2003

[2] Peppler, K. & Kafai, Y. B., Collaboration, Computation, and Creativity: Media Arts Practices in Urban Youth Culture. In C. Hmelo- Silver & A. O'Donnell (Eds.), In Proc. Computer Supported Collaborative Learning, New Brunswick, NJ, USA, 2007

[3] Repenning, A., Excuse me, I need better AI! Employing Collaborative Diffusion to make Game AI Child's Play. In Proc. ACM SIGGRAPH Video Game Symposium, Boston, MA, USA, ACM Press, 2006.

[4] Sturtevant, N. R., Hoover, H. J., Schaeffer, J., Gouglas, S., Bowling, M. H., Southey, F., Bouchard, M., and Zabaneh, G. 2008. Multidisciplinary students and instructors: a second-year games course. In proc 39th SIGCSE Technical Symposium on Computer Science Education, Portland, OR, USA, 2008.

[5] Squire, K., Video games in education. International Journal of Intelligent Simulations and Gaming, (2) 1. 2003.

[6] Repenning, A. and A. Ioannidou 2005. What makes End-User Development Tick? 13 design guidelines. End-User Development. F. Patern&Mac247; and V. Wolf. Dordrecht, Kluwer

[7] Basawapatna A., Koh, K.H., Repenning, A. 2010. Using Scalable Game Design to Teach Computer Science From Middle School to Graduate School. In Proceedings of ITICSE '10 (Bilkent, Ankara, Turkey, June 26-30, 2010).

[8] Repenning, A., "AgentSheets®: an Interactive Simulation Environment with End-User Programmable Agents," Interaction 2000, Tokyo, Japan, 2000.

[9] Ruthmann, A., Heines, J. M., Greher, G. R., Laidler, P., and Saulters, C. 2010. Teaching computational thinking through musical live coding in scratch. In Proceedings of the 41st ACM Technical Symposium on Computer Science Education Milwaukee, Wisconsin, USA, March 10 - 13, 2010. SIGCSE '10.

[10] Repenning, A., Webb, D., and Ioannidou, A., "Scalable Game Design and the Development of a Checklist for Getting Computational Thinking into Public Schools", Proc. SIGCSE' 10, ACM Press, WI, USA, 2010.

[11] Barr, D., Harrison, J., and Conery, L. (2011) Computational Thinking: A Digital Age Skill For Everyone. ISTE Learning and Leading, 38(6), pp. 20-23. March/April 2011.

[12] Wing, J. M., "Computational Thinking," Communications of the ACM, 49(3), pp. 33-35, March 2006.

[13] Wing, J.M., "Computational Thinking and Thinking About Computing," Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences. 2008 vol. 366 (1881) pp. 3717.

[14] National Academy of Sciences on Computational Thinking. Report of a Workshop on The Scope and Nature Computational Thinking, National Academies Press, 2010.

[15] Peter B. Henderson. 2009. Ubiquitous Computational Thinking. Computer 42, 10 (October 2009), 100-102.

[16] Repenning, A., Ioannidou, A., & Zola, J. (2000). AgentSheets: End-User Programmable Simulations. Journal of Artificial Societies and Social Simulation, 3(3).

[17] Repenning, A. and C. Lewis 2005. Playing a Game: The Ecology of Designing, Building and Testing Games as Educational Activities. ED-Media 2005, World Conference on Educational Multimedia, Hypermedia & Telecommunications, Montreal, Canada, Association for the Advancement of Computing in Education.

[18] Presenter: Len Scrogan, 2010. Retrieved March 27, 2011, from TEDx DenverED: http://tedxdenvered.com/blog/2010/06/19/presenter-len-scrogan-2.

[19] Amylase. Retreived March 27, 2011, from Indiana University Office of Science Outreach: http://www.indiana.edu/~oso/animations/amylase.html.

[20] Amylase. Retrieved March 27, 2011, from Wikipedia: http://en.wikipedia.org/wiki/Amylase.

[21] Vuorisal, T., Arjamaa, O. *American Scientist*. March–April 2010.

[22] Shiflet, A., Shiflet, G. Introduction to Computational Science. Princeton University Press, Princeton, New Jersey, 2006, pp. 224-235.

[23] Gray L., Thomas, N., Lewis, L., Tice, P. Teachers' Use of Educational Technology in U.S. Public Schools: 2009, First Look. US Department of Education Report, National Center for Education Statistics.

[24] Computer Science via Interactive Journalism (2009). Retrieved March 27, 2011, NSF Press Release 09-239: http://www.nsf.gov/news/news_summ.jsp?cntn_id=116073 .

[25] Learning Computer Science From Scratch (2009). Retrieved March 27, 2011, NSF Press Release 09-235: http://www.nsf.gov/news/news_summ.jsp?cntn_id=116070.

[26] Mathematics>>High School: Modeling>> Introduction. Retrieved March 27, 2011, Common Core: State Standards Initiative: http://www.corestandards.org/the-standards/mathematics/high-school-modeling/introduction/

[27] Basawapatna, A., Koh, K.H., Repenning, A., Webb, D.C., and Marshall, K.S., "Recognizing Computational Thinking Patterns", Proc. SIGCSE' 11, ACM Press, TX, USA, 2010.

[28] Valadez, JR., Duran. R. Redefining the digital divide: Beyond access to computers and the Internet. High School Journal, 2007 vol. 90 (3) pp. 31.

[29] Repenning, A. 2005. Inflatable Icons: Diffusion-based Interactive Extrusion of 2D Images into 3D Models. The Journal of Graphics Tools 10(1): 1-15.

[30] Repenning, A., Collaborative Diffusion: Programming Antiobjects. in OOPSLA 2006, ACM SIGPLAN International Conference on Object-Oriented Programming Systems, Languages, and Applications, (Portland, Oregon, 2006), ACM Press.

[31] Alexander, C., Ishikawa, S., Silverstein, M. A Pattern Language: Towns, Buildings, Construction. Oxford University Press, New York, New York, 1977.

[32] Gamma, E., Helm, R., Johnson, R., Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Boston, Massachusetts, 1995.

[33] Michotte, A. The Perception of Causality. Metheun & CO LTD, London, England, 1963.

[34] Bjork, S., Holopianen, J. Patterns in Game Design. Charles River Media Game Development, Boston, Massachusetts, 2004.

[35] Gestwicki, P., Sun, F. Teaching Design Patterns Through Computer Game Development. ACM Journal of Educational Resources and Computing. 2008 vol. 8 (1).

[36] Fischer, G., Lemke, A. Construction Kits and Design Environments: Steps Toward Human Problem-Domain Communication. Human Computer Interaction. 1987-1988 vol. 3 pp. 179-222.

[37] Iannidou, A., Repenning, A., Lewis, C., Cherry, G., Rader, C. Making Constructionism Work In The Classroom. International Journal of Computers for Mathematical Learning. 2003 vol. 8 pp. 63-108.

[38] Kelleher, C. Motivating Programming: Using Storytelling to Make Computer Programming Attractive To Middle School Girls. Thesis Submission School of Computer Science, Carnegie Melon University. 2006.

[39] Podelfsky, N., Perkins, K., Adams, W. Factors Promoting Engaged Expoloration With Computer Simulations. Physical Review Special Topics- Physics Education Research. 2010.

[40] Adams, W., Paulson, A., Wieman, C. What Levels of Guidance Promote Engaged Exploration With Interactive Simulations. PERC Proceedings. 2009.

[41] Klopfer, E., Yoon, S., Um, T. Teaching Complex Dynamic Systems to Young Students With Starlogo. Journal of Computers In Mathematics. 2005.

[42] Klopfer, E., Scheintaub, H., Huang, W., Wedel, D., Roque, R. The Simulation Cycle: Combining Games, Simulations, Engineering, and Science using Starlogo TNG. E-Learning. 2009 vol 6 (1) pp. 71-96.

[43] Davis, R. Magic Paper: Sketch-Understanding Research. Computer. 2007 vol. 40 (9) pp. 34-41.

[44] Kelleher, C. Pausch, R. Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers. ACM Computing Surveys. 2005 vol. 37 (2) pp. 83-137.

[45] Galas, C. School Squeaking. Squeak News. 2001 vol. 1 (4).

[46] Maloney, J., Burd, L., Kafai, Y., Rusk, N., Silverman, B., Resnick, M. Scratch: A Sneak Preview. Second International Conference on Creating, Connecting and Collaborating through Computing. 2004 pp. 104-109.

[47] Hill, B.M., Monroy-Herández, A., Olson, K.R. Responses to Remixing On a Social Media Sharing Website. AAAI Proceedings. 2010.

[48] Brennan, K., Monroy- Herández, A., Resnick, M. Scratch: Creating and Sharing Interactive Media. In Proceedings CSCL. 2009 vol. 2.

[49] Kolling, M., Henriksen, P. Game Programming in Introductory Courses With Direct State Manipulation. In Proceedings of ITiCSE. 2005.

[50] Detlef, R., Ludwig, J., Meyer, S., Rehder, C., Schirmer, I. Introduction to Business Informatics With Greenfoot Using the Example of Airport Baggage Handling. In Proceedings of Koli Calling. 2010 pp. 68-69.

[51] Boutin, P. The 8-Year-Old Programmer. The New York Times Sunday Magazine. September 16, 2010, pp. MM62.

[52] Koh, K., Basawapatna, A., Bennet, V., Repenning, A. Towards The Automatic Recognition Of Computational Thinking for Adaptive Visual Language Learning. In Proceedings of VL/HCC. 2010.

[53] Basawapatna, A., Repenning, A. Visualizing Student Game Design Project Similarities. In Proceedings of Diagrammatic Representation and Inference. 2010.

[54] Chappuis, J., Stiggins, R., Chappius, S., Arter, J. Classroom Assessment For Student Learning: Doing It Right—Using It Well. Pearson, Upper Saddle River, New Jersey, 2011.

[55] Furtak, E. Formative Assessment For Secondary Science Teachers. Corwin, Thousand Oaks, California, 2009.

[56] Boulder Valley School District. Middle School Life Science Curriculum Essentials Document. Boulder Valley School District Department of Curriculum And Instruction May 2009.

[57] John Stasko, Albert Badre, and Clayton Lewis. 1993. Do algorithm animations assist learning?: an empirical study and analysis. In

Proceedings of the INTERACT '93 and CHI '93 conference on Human factors in computing systems (CHI '93). 1993.

[58] Sivilotti and Laugel. Scratching the surface of advanced topics in software engineering: a workshop module for middle school students. Proceedings of the 39th SIGCSE technical symposium on Computer science education (2008) pp. 291-295

[59] Pulimood and Wolz. Problem solving in community: a necessary shift in cs pedagogy. Proceedings of the 39th SIGCSE technical symposium on Computer science education (2008) pp. 210-214

[60] Repenning, A., & Ambach, J.,"Visual AgenTalk: Anatomy of a Low Threshold, High Ceiling End User Programming Environment," *Department of Computer Science, University of Colorado Tech Report # CU-CS-802-96*, January, 1996.

[61] Repenning, A., Making Programming more Conversational,in Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC '11, (Pittsburgh, PA, USA Sept. 18-22, 2011), IEEE Computer Society, Los Alamitos, CA.

[62] The impact of web 2.0 technologies in the classroom. State of Victoria: Department of Education and Early Childhood Development, December 2009.

[63] Bennett, V., Koh, K. H., Repenning, A. Can Learning Acquisition be Computed?, IEEE International Symposium on Visual Languages and Human-Centric Computing 2011, Pittsburgh, PA, USA, September 18-22, 2011

[64] Brand, Cathy, and Cyndi Rader. "How does a visual simulation program support students creating science models?." *Visual Languages, 1996. Proceedings., IEEE Symposium on*. IEEE, 1996.

[65] Minar, Nelson, et al. "The swarm simulation system: A toolkit for building multi-agent simulations." Santa Fe, NM, USA: Santa Fe Institute, 1996.

[66] Kelleher, C., Pausch,R. , and Kiesler, S. "Storytelling alice motivates middle school girls to learn computer programming." *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 2007.

[67] Ioannidou, A., Bennett, V., Repenning, A., Koh, K., Basawapatna, A., Computational Thinking Patterns. Paper presented at the 2011

Annual Meeting of the American Educational Research Association (AERA) in the symposium "Merging Human Creativity and the Power of Technology: Computational Thinking in the K-12 Classroom", New Orleans, April 8-12, 2011

[68] Marshall, K. S. (2011). Was that CT? Assessing computational thinking patterns through videobased prompts. Paper presented at the American Educational Research Association, New Orleans, LA. http://www.eric.ed.gov/ERICWebPortal/contentdelivery/servlet/ERICServlet?accno=ED518514

[69] Gilbert, N., & Bankes, S. (2002). Platforms and methods for agent-based modeling. *Proceedings of the National Academy of Sciences of the United States of America*, *99*(Suppl 3), 7197-7198.

[70] Gilbert, N., & Bankes, S. (2002). Platforms and methods for agent-based modeling. *Proceedings of the National Academy of Sciences of the United States of America*, *99*(Suppl 3), 7197-7198.

[71] Stephen Cooper. 2010. The Design of Alice. *Trans. Comput. Educ.* 10, 4, Article 15 (November 2010), 16 pages.

[72] Colleen M. Lewis. 2010. How programming environment shapes perception, learning and goals: logo vs. scratch. In *Proceedings of the 41st ACM technical symposium on Computer science education* (SIGCSE '10).

[73] Brand. Lowering Barriers to Interaction: Programming without Code. Visual Languages and Human-Centric Computing, 2007. VL/HCC 2007. IEEE Symposium on (2007) pp. 250-251

[74] Reed, D. A., Bajcsy, R., Fernandez, M. A., Griffiths, J. M., Mott, R. D., Dongarra, J., & Ponick, T. L. (2005). *Computational Science: Ensuring America's Competitiveness*. PRESIDENT'S INFORMATION TECHNOLOGY ADVISORY COMMITTEE ARLINGTON VA.

[75] Kahn. Drawings on napkins, video-game animation, and other ways to program computers. Communications of the ACM (1996) vol. 39 (8) pp. 49-59

[76] Utting, I., Cooper, S., Kölling, M., Maloney, J., & Resnick, M. (2010). Alice, greenfoot, and scratch--a discussion. *ACM Transactions on Computing Education (TOCE)*, *10*(4)

[77] Repenning, A., & Ioannidou, A. (2007, April). X-expressions in XMLisp: S-expressions and extensible markup language unite. In *Proceedings of the 2007 International Lisp Conference* (p. 23). ACM.

[78] Repenning, A., & Citrin, W. (1993). Agentsheets: Applying Grid-Based Spatial Reasoning to Human- Computer Interaction. In 1993 IEEE Workshop on Visual Languages, Bergen, Norway: IEEE Computer Society Press.

[79] G Fischer. 1987. An object-oriented construction and tool kit for human-computer communication. *SIGGRAPH Comput. Graph.* 21, 2 (April 1987), 105-109.

[80] Olson, C. K. (2010). Children's motivations for video game play in the context of normal development. *Review of General Psychology*, *14*(2), 180.

APPENDIX A: Summary of AgentCubes Conditions and Actions

The following is the Conditions palettes in AgentCubes, with descriptions, used for this thesis (more conditions may have been added to AgentCubes post thesis). It includes 2 Figures. The first figure presents the "basic conditions" palette and the second figure presents the "keyboard", "attributes", and "camera control" palettes. Note that all conditions and actions can be negated in AgentCubes.

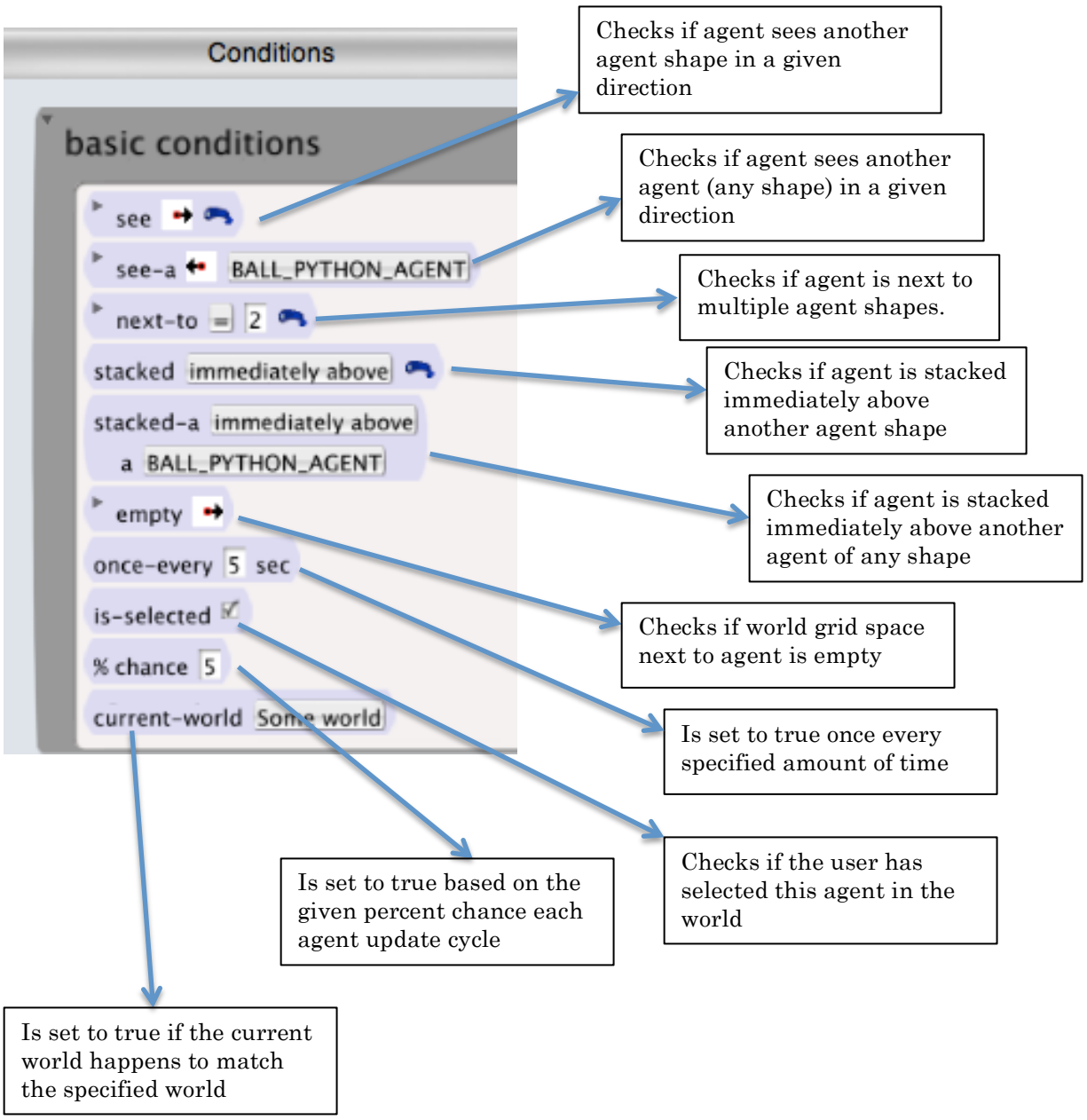


Figure 120: Basic Conditions Palette In AgentCubes

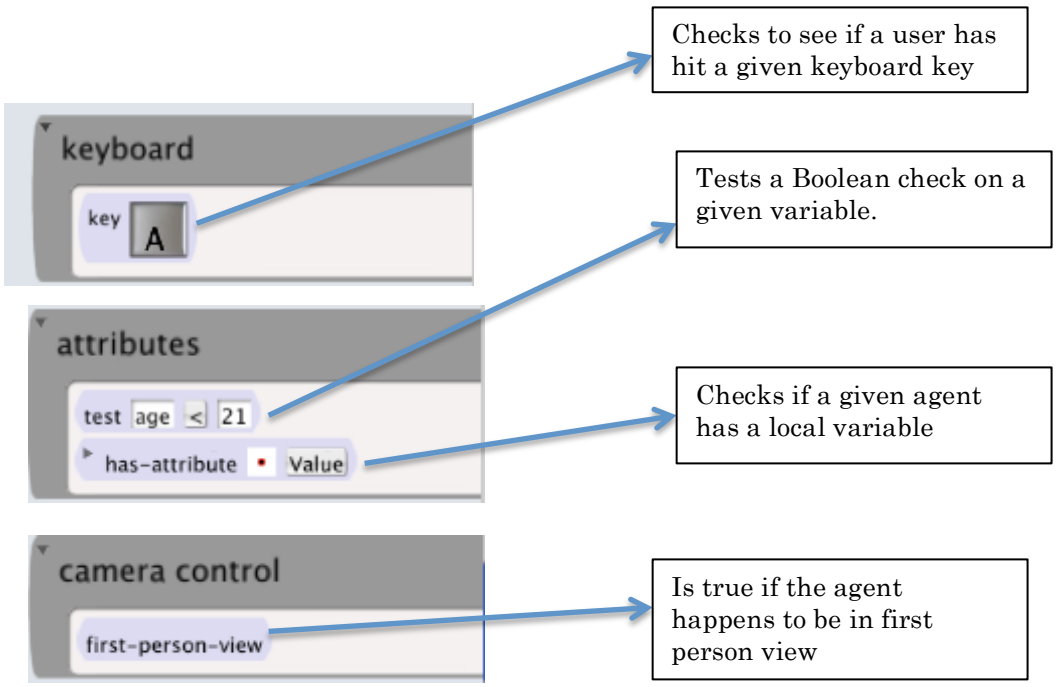


Figure 121: Keyboard, Attributes, And Camera Control Condition Palettes

The following figures present the AgentCubes Action Palette. It should be noted that since there are many actions, I will only be presenting the ones relevant to this thesis; however, more information on AgentCubes behaviors can be found at www.agentsheets.com.

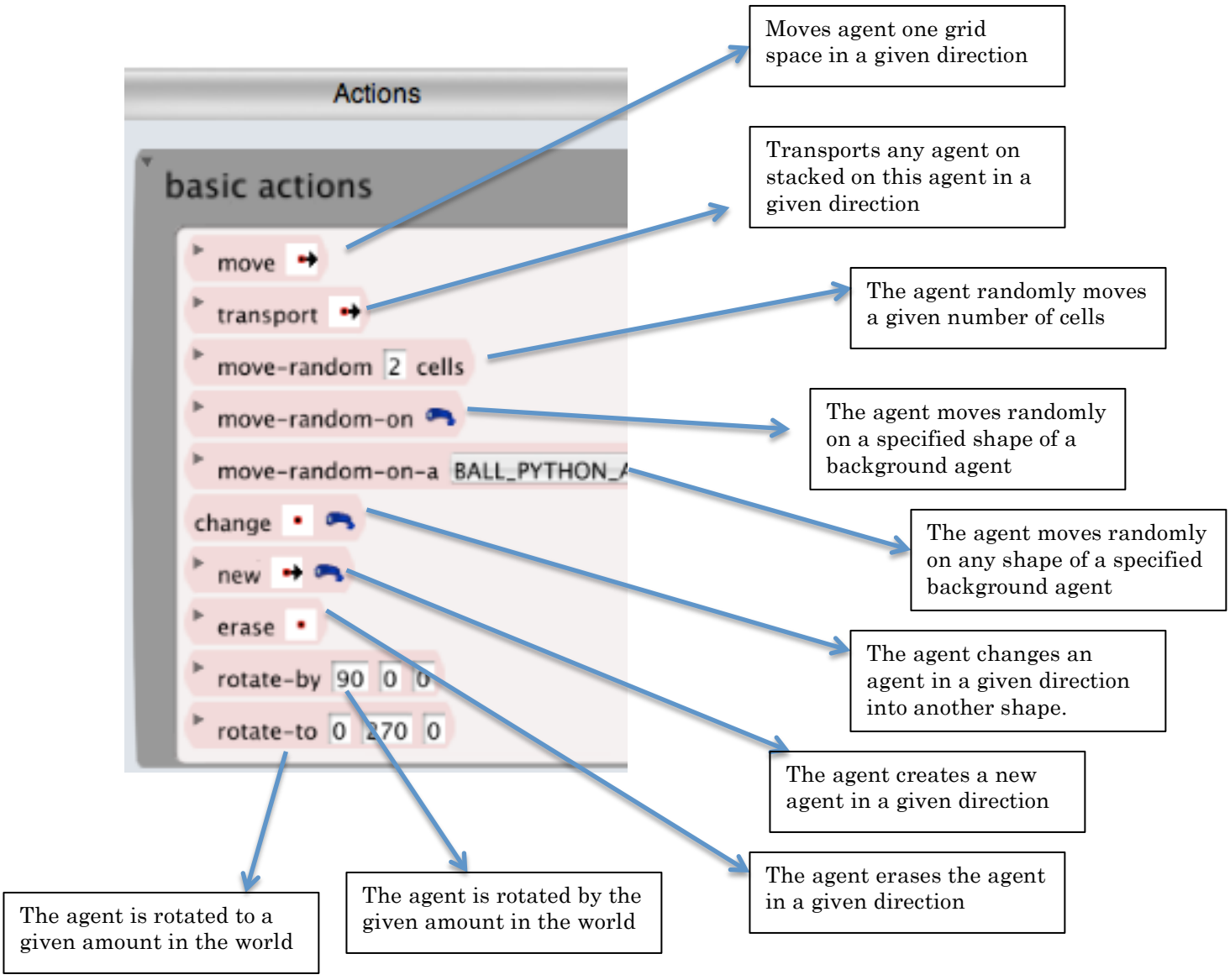


Figure 122: Basic Actions Palette In AgentCubes

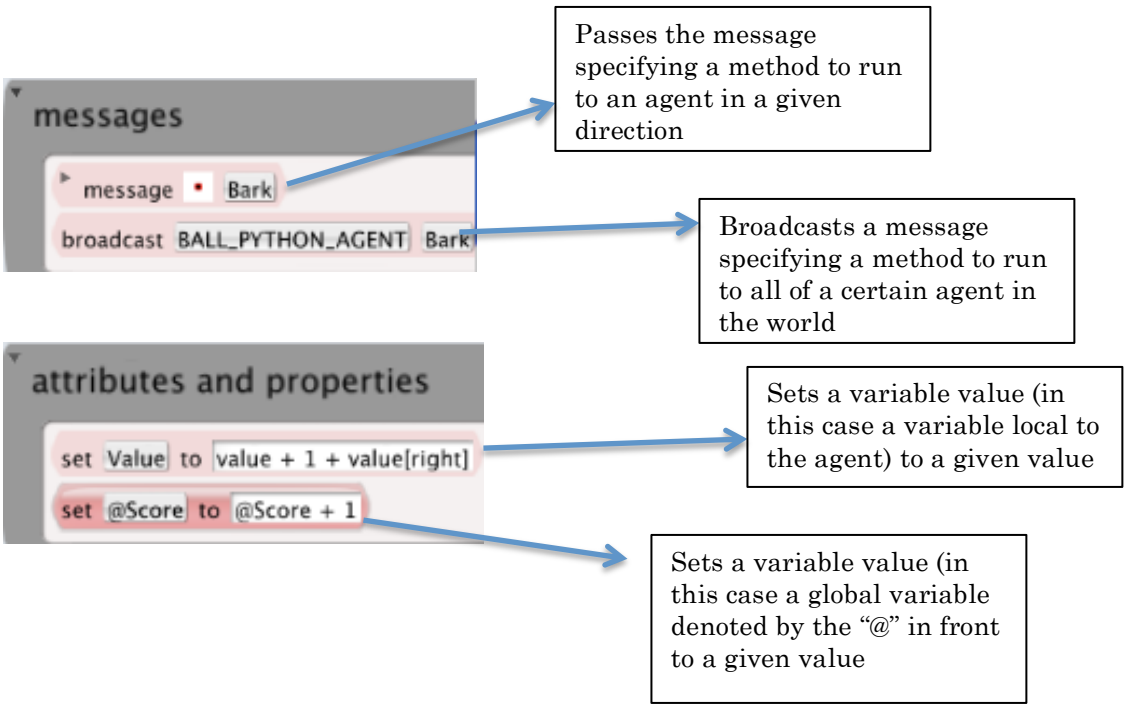


Figure 123: Message And Attributes/Properties Action Palettes In AgentCubes

APPENDIX B: Summary and Discussion Of The Simulation Creation Toolkit Windows And Interacticons

This section will present all the windows of the Simulation Creation Toolkit with a brief description. It will also describe each pattern's interacticons; these will be described in each pattern window section. This will be followed by a brief discussion of the windows and interacticons in the Simulation Creation Toolkit.

It should be noted that though this Appendix shows the generic disk versions of the interacticons, as pointed out in Chapter 3, once the user selects an agent in the pattern window, the generic disk corresponding to that agent changes into the agent itself. Therefore, after agent selection, the agents of the simulation, rather than the generic interacticon objects, act out the pattern represented in the interacticon. Furthermore, all interacticons loop back to the beginning of the animation when done unless otherwise stated.

B.1 The Simulation Construction Kit Window

The Simulation Construction Kit Window looks as follows.

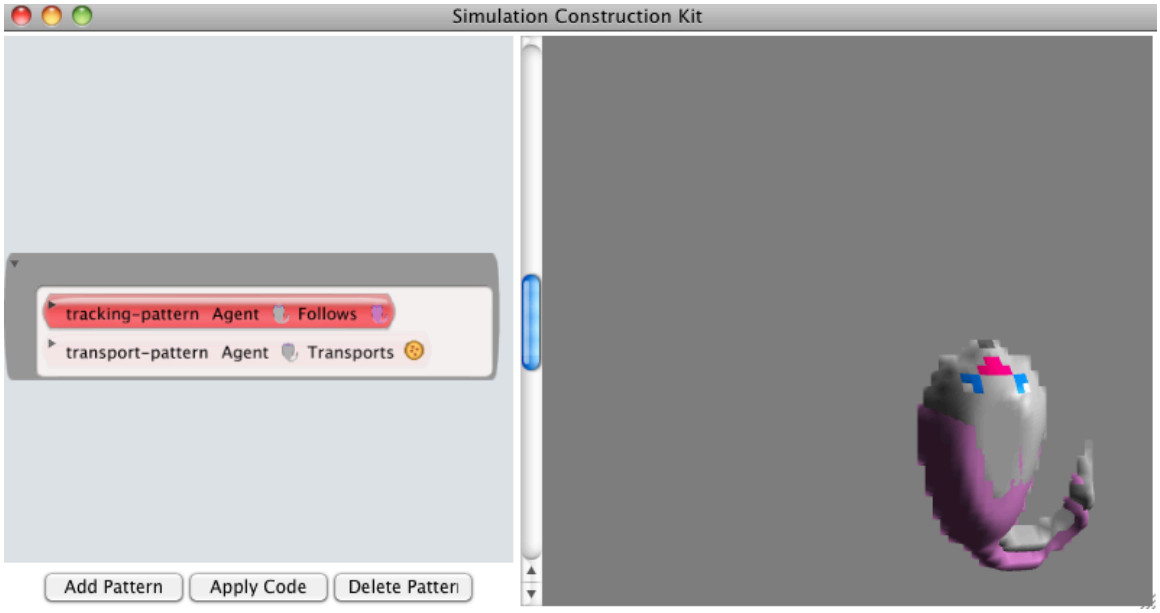


Figure 124: Example Simulation Construction Kit Window

The left side of this window shows all the implemented patterns (in this case the Tracking and Transport Pattern), and the right side displays an interacticon corresponding to the selected left side pattern (in this case the Tracking Pattern). The "Add Pattern" button on the bottom left launches the Pattern Picker Window. The "Apply Code" button in the middle serves no purpose and will be taken out in subsequent versions of the Simulation Creation Toolkit. Finally, the "Delete Pattern" on the bottom right deletes the selected pattern and its corresponding AgentCubes code.

B.2 The Pattern Picker Window

The Pattern Picker Window looks as follows.

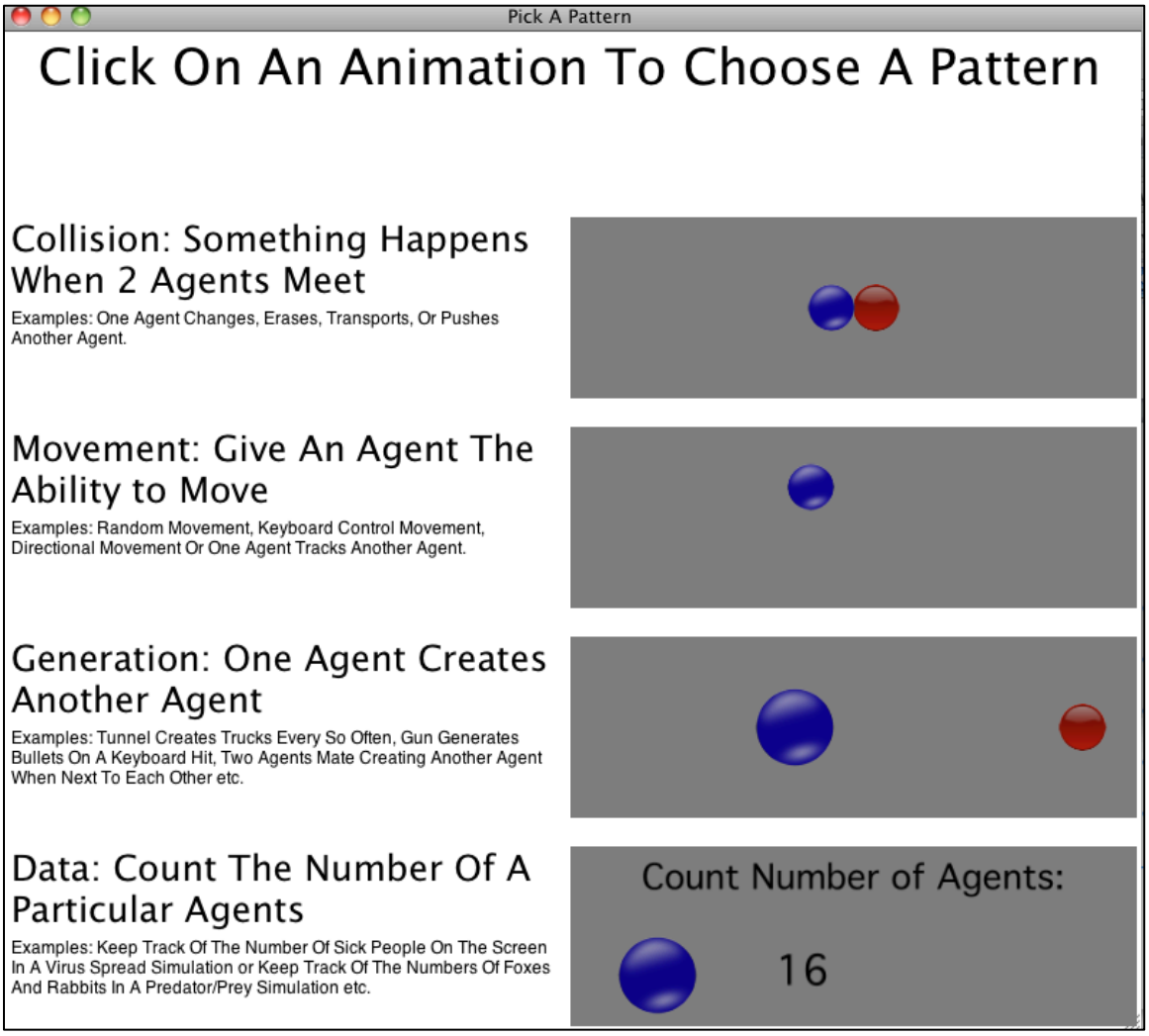


Figure 125: The Pattern Picker Window

The Pattern Picker Window has descriptions of Pattern Categories on the left side and a corresponding representative animation on the right side. The first pattern category from the top refers to Collision Patterns. These include the Change Pattern, the Absorb Pattern, the Transport Pattern and the Push Pattern. The corresponding animation is a collision interacticon wherein the red and the blue disks come together. Clicking on this interacticon launches the Collision Picker Window.

The second pattern category refers to the Movement Pattern. This category includes the Random Movement Pattern, the Tracking Pattern, the

Keyboard Control Movement Pattern, and the Directional Movement Pattern. The interacticon used for this category is the Random Movement Pattern interacticon that will be described later. Clicking on this interacticon launches the Movement Picker Window.

The third category refers to the Generate Pattern (the Generate Pattern is its own category see 3.2). Thus, the corresponding interacticon is the Generate Pattern interacticon that will be described later. Clicking on this interacticon launches the Generate Pattern Window.

The fourth category is the Data Pattern (like the Generate Pattern, the Data Pattern is its own category). The corresponding interacticon belongs to the Data Pattern and will be described later.

B.3 The Collision Picker Window

The Collision Picker Window looks as follows. It is organized in a similar fashion to the Pattern Picker window (see Appendix B.2) with pattern descriptions on the left and corresponding interacticons on the right.

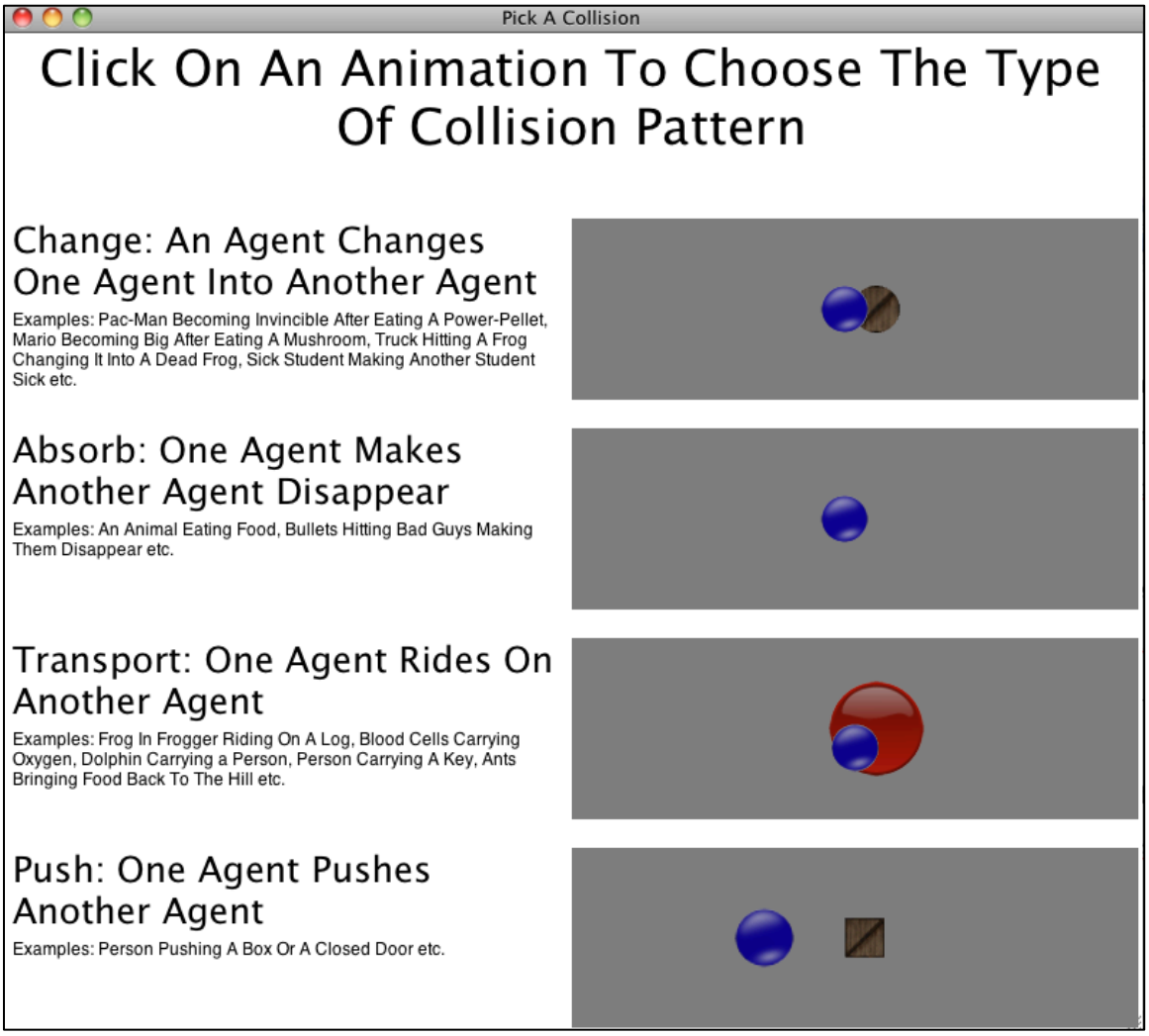


Figure 126: The Collision Picker Window

The first pattern from the top shown in The Collision Picker Window is the Change Pattern. Clicking on this pattern's interacticon opens the Change Pattern Window. The second pattern from the top is the Absorb Pattern. Clicking on this pattern's interacticon opens the Absorb Pattern Window. The third pattern from the top is the Transport Pattern. Clicking on this pattern's interacticon opens the Transport Pattern Window. Finally, the bottom pattern in this window is the Push Pattern. Clicking on this pattern's interacticon opens the Push Pattern Window.

B.4 The Change Pattern Window

The Change Pattern Window looks as follows.

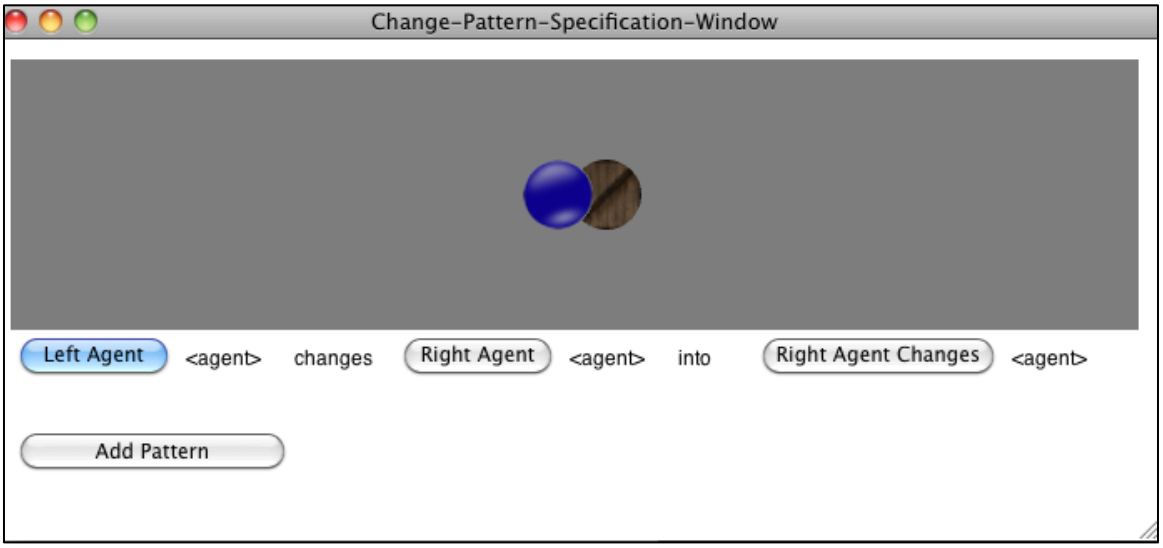


Figure 127: The Change Pattern Window

The Change Pattern Window has a button that lets the user specify the "Left Agent" which refers to the Changer Agent in the interacticon and the pattern. The "Right Agent" button enables the user to specify the Agent To Be Changed. The "Right Agent Changes" button allows the user to specify the Change Into Agent.

The following is three sequential frames, from left to right, of the Change Pattern interacticon.

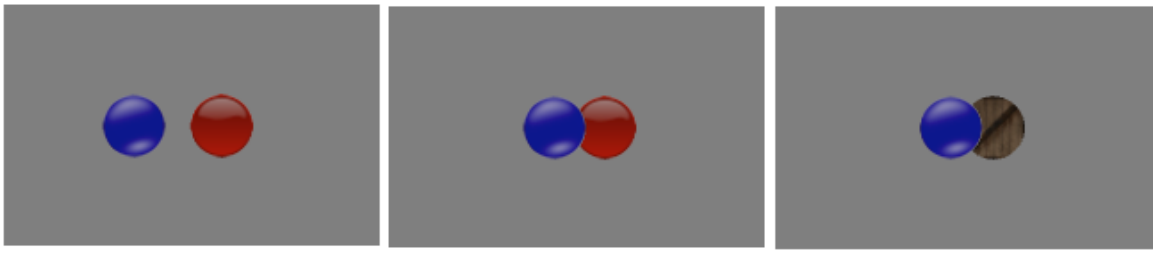


Figure 128: Three Sequential Frames Of The Change Pattern Interacticon

The left blue disk in the Change Pattern interacticon moves right and collides with the right red disk, which is moving left, and when this collision happens, the red disk changes into a wood textured disk.

B.5 The Absorb Pattern Window

The Absorb Pattern Window looks as follows.

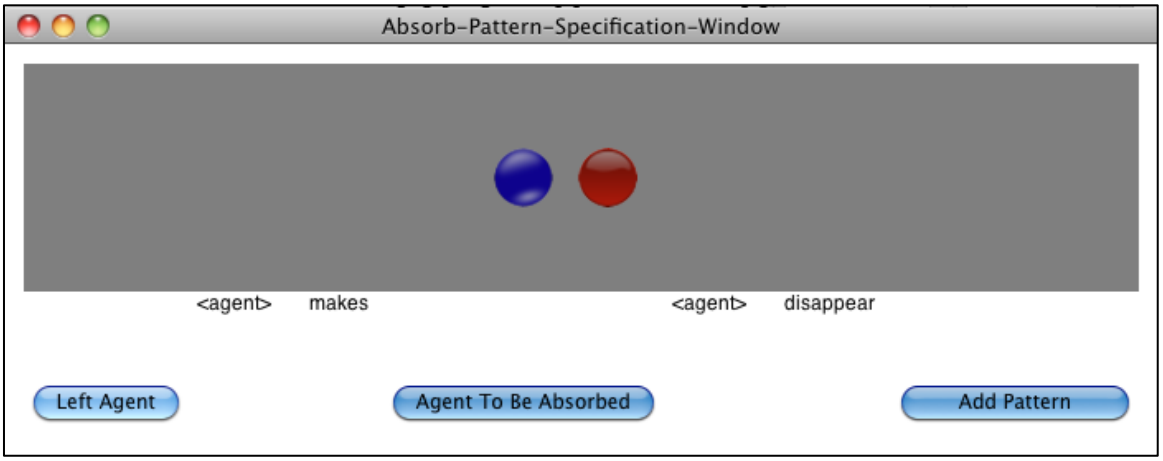


Figure 129: The Absorb Pattern Window

The "Left Agent" button specifies the Absorber Agent. The "Agent To Be Absorbed" button specifies that Absorbed Agent.

The following is three sequential frames, from left to right, of the Absorb Pattern interacticon.

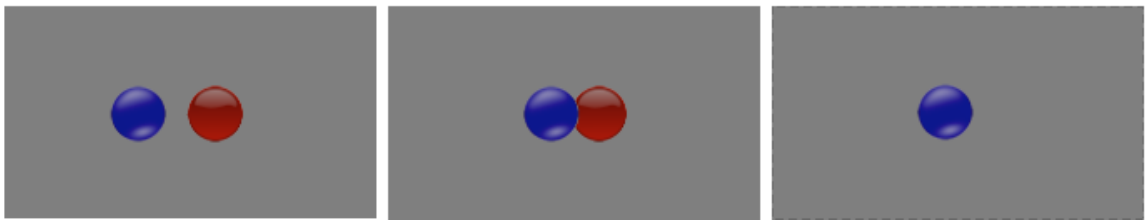


Figure 130: Three Sequential Frames Of The Absorb Pattern Interacticon

In the Absorb Pattern interacticon, the blue disk on the left moves right colliding with the red disk on the right, which is moving left, and upon collision, the red disk disappears.

B.6 The Transport Pattern Window

The Transport Pattern Window looks as follows.

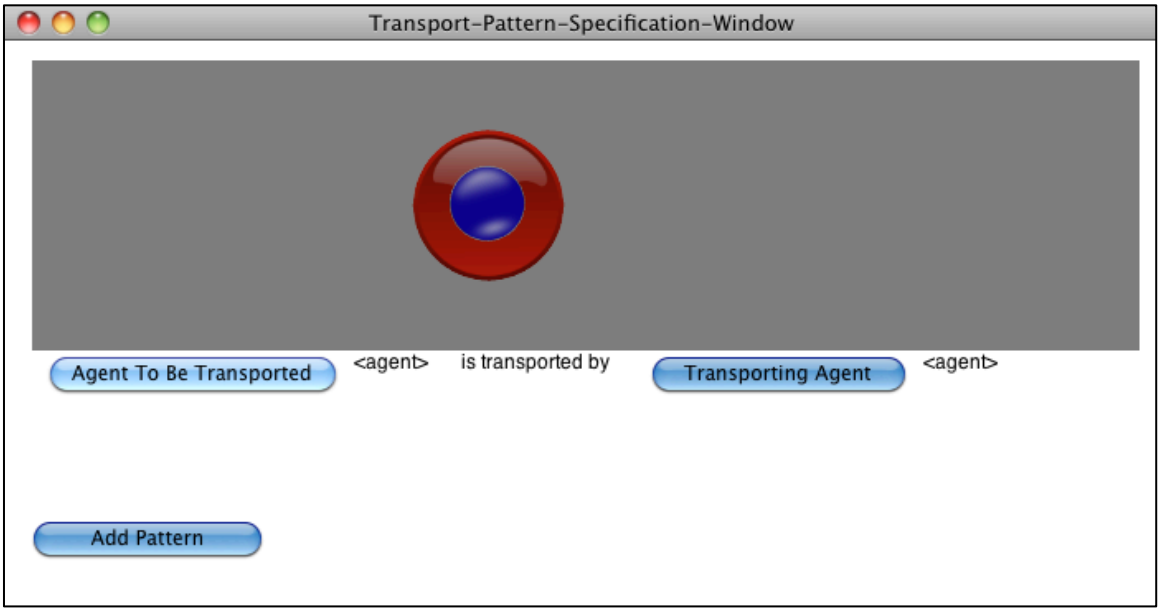


Figure 131: The Transport Pattern Window

The Transport Pattern Window allows the user to select the "Agent To Be Transported" and the "Transporting Agent."

The following are three sequential frames, from left to right, of the Transport Pattern interacticon.

Figure 132: Three Sequential Frames Of The Transport Pattern Interacticon

In the Transport Pattern interaction, the red transporting disk moves from left to right. The blue transported disk starts at the bottom of the screen and "jumps up onto" the red disk as it passes. At this point the blue disk and the red disk move together to the left with the same velocity.

B.7 The Push Pattern Window

The following picture depicts the Push Pattern Window.

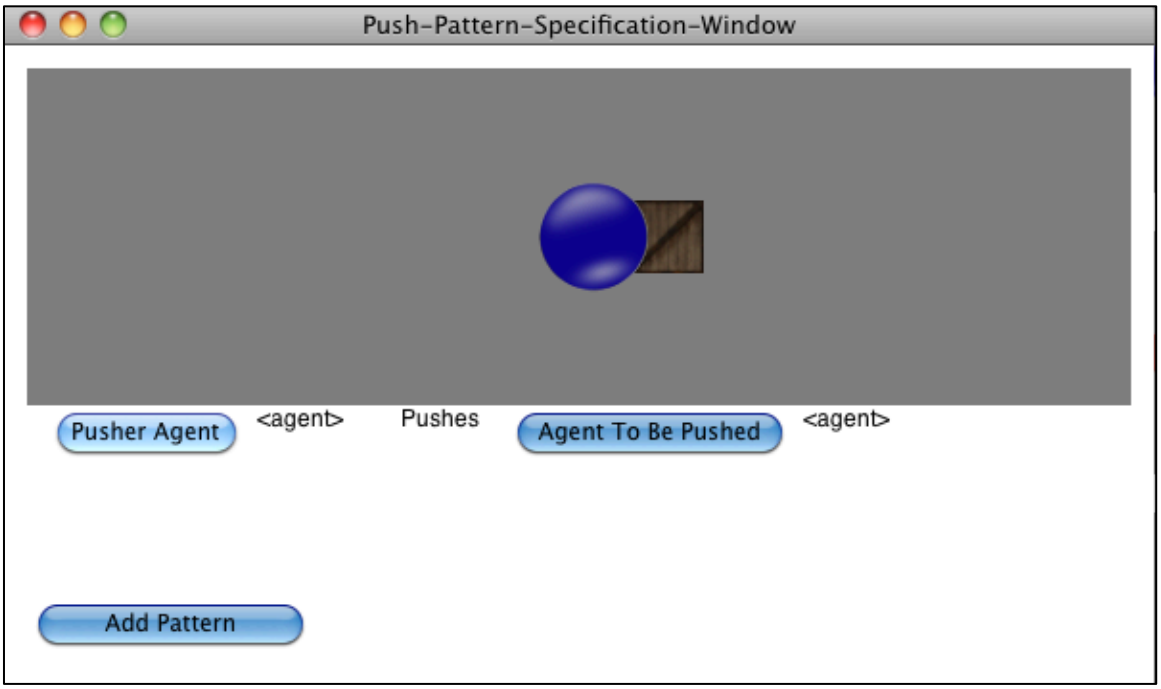


Figure 133: The Push Pattern Window

The Push Pattern Window enables a user to select a "Pusher Agent" and an "Agent To Be Pushed."

The following depicts three sequential frames, from left to right, of the Push Pattern interacticon.

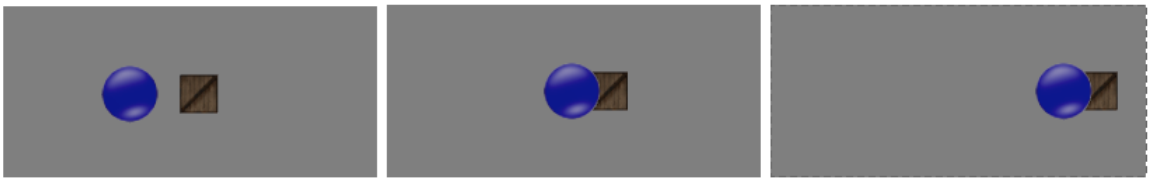


Figure 134: Three Sequential Frames Of The Push Pattern Interacticon

In the Push Pattern interacticon, a blue disk moves to the right and when it collides with a stationary box, the blue disk starts pushing the box to the right (ie: the box starts moving to the right with the same velocity as the blue disk).

B.8 The Movement Picker Window

The following figure depicts the Movement Picker Window. It is organized similar to the Collision Picker window with descriptions on the left and corresponding interacticons on the right.

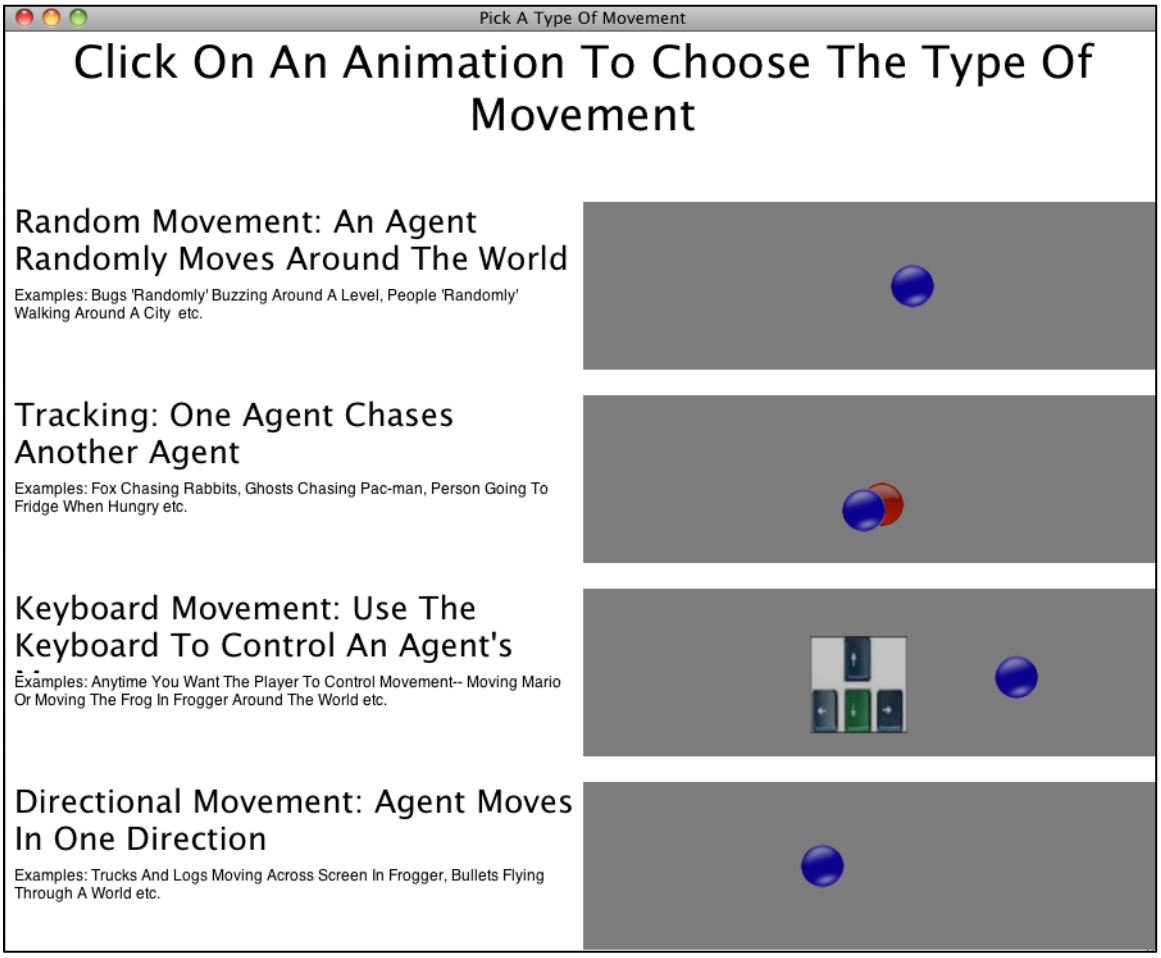


Figure 135: The Movement Picker Window

The first pattern from the top shown in The Movement Picker Window is the Random Movement Pattern. Clicking on this pattern's interacticon opens the Random Movement Pattern Window. The second pattern from the top is the Tracking Pattern. Clicking on this pattern's interacticon opens the Tracking Pattern Window. The third pattern from the top is the Keyboard Control Movement Pattern. Clicking on this pattern's interacticon opens the Keyboard Control Pattern Window. Finally, the bottom pattern in this window is the Directional Movement Pattern. Clicking on this pattern's interacticon opens the Directional Movement Pattern Window.

B.9 The Random Movement Pattern Window

The following figure depicts the Random Movement Pattern Window.

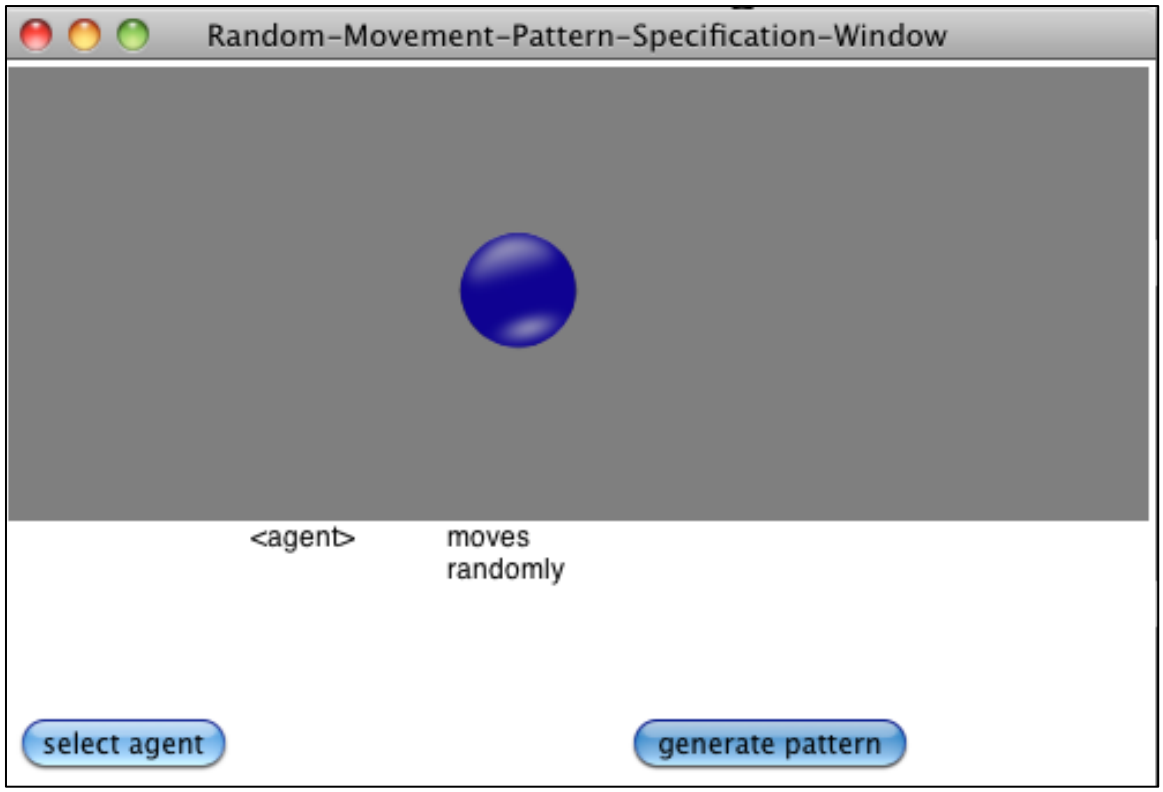


Figure 136: The Random Movement Pattern Window

The Random Movement Pattern Window has a "select agent" button that enables a user to specify the Randomly Moving Agent. Showing frames of the Random Movement Pattern interacticon are not descriptive so instead we will describe the movement. The blue disk agent shown in The Random Movement Pattern Window chooses a place to move using a random number modulated to the height and width dimensions of the grey box in the Random Movement Pattern Window. Once it reaches to this spot, the disk chooses a new random target to move towards. Therefore, unlike most other

interacticons, the Random Movement Pattern interacticon does not loop the same animation.

B.10 The Tracking Pattern Window

The following picture depicts the Tracking Pattern Window.

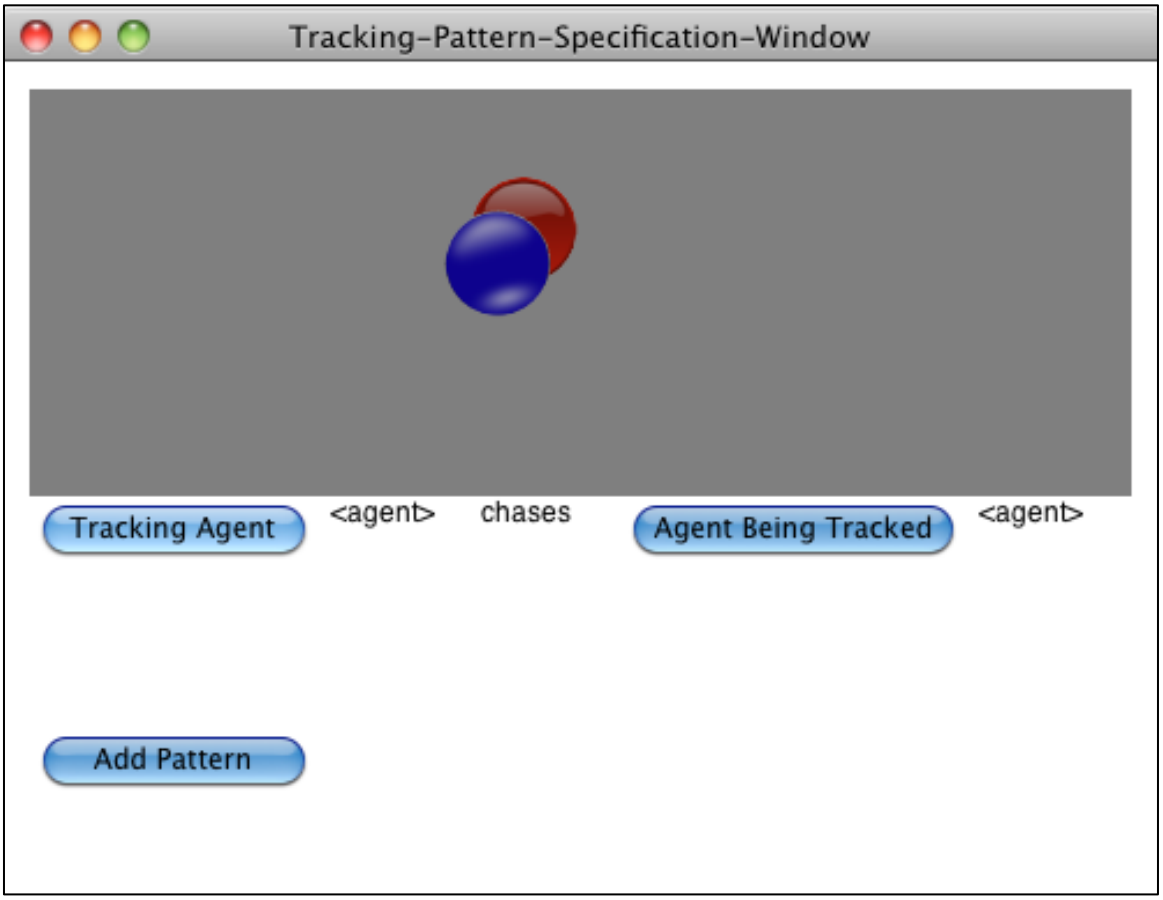


Figure 137: The Tracking Pattern Window

The Tracking Pattern Window enables a user to select a "Tracking Agent" and an "Agent Being Tracked." It is hard to see this tracking in a limited number of frames so a description of this interacticon will be used instead. The blue disk agent, similar to the Random Movement Pattern interacticon,

randomly picks a target spot in the grey portion of the Tracking Pattern Window to travel to. Once the blue disk agent reaches that spot, it picks a new target location. The red disk agent uses the blue disk agent's current location as its target. The red disk agent, however, moves slightly slower than the blue disk agent leading to the perception that the red disk agent is chasing the blue disk agent. Like the Random Movement Pattern interacticon, the Tracking Pattern interacticon does not loop the animation.

B.11 The Keyboard Control Movement Pattern Window

The following picture depicts the Keyboard Control Movement Pattern Window.

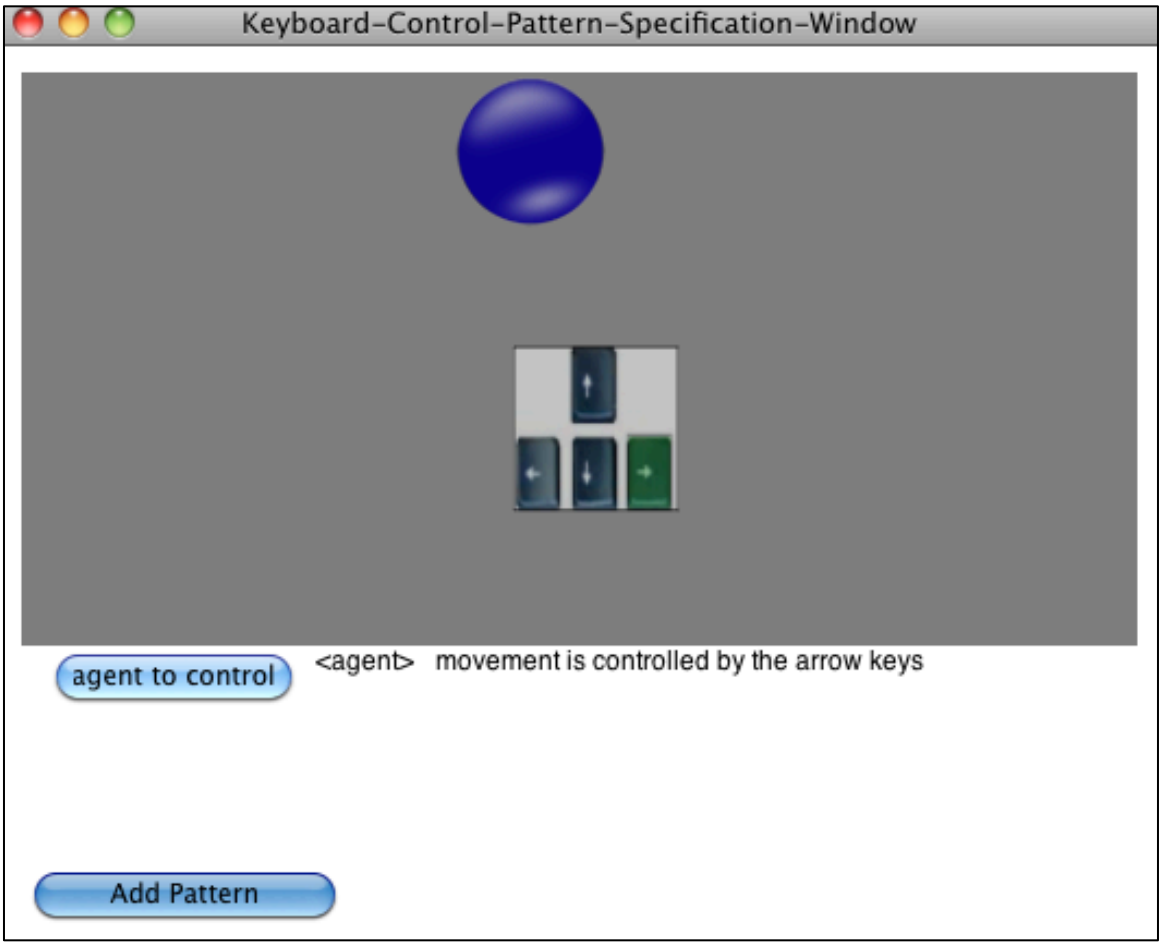


Figure 138: The Keyboard Control Movement Pattern Window

The Keyboard Control Movement Pattern Window enables a user to pick an agent move via keyboard keys by using the "agent to control" button.

The following picture depicts four non-sequential frames of the Keyboard Control Pattern interacticon. The blue arrows refer to where the disk is moving in a given frame.

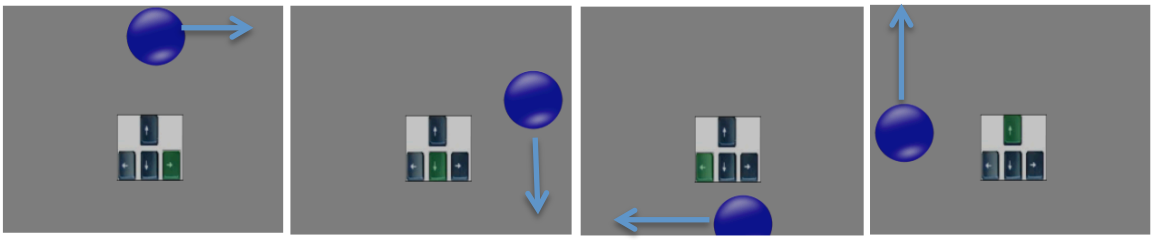


Figure 139: Four Non-Sequential Frames Of The Keyboard Control Pattern Interacticon

In the Keyboard Control Pattern Interacticon, the blue disk moves in a square-wise fashion around a picture of arrow keys on a keyboard. As the disk moves, the keyboard arrow buttons, corresponding to the direction the disk is moving at that time, get highlighted in green.

B.12 The Directional Movement Pattern Window

The following picture depicts the Directional Movement Pattern Window.

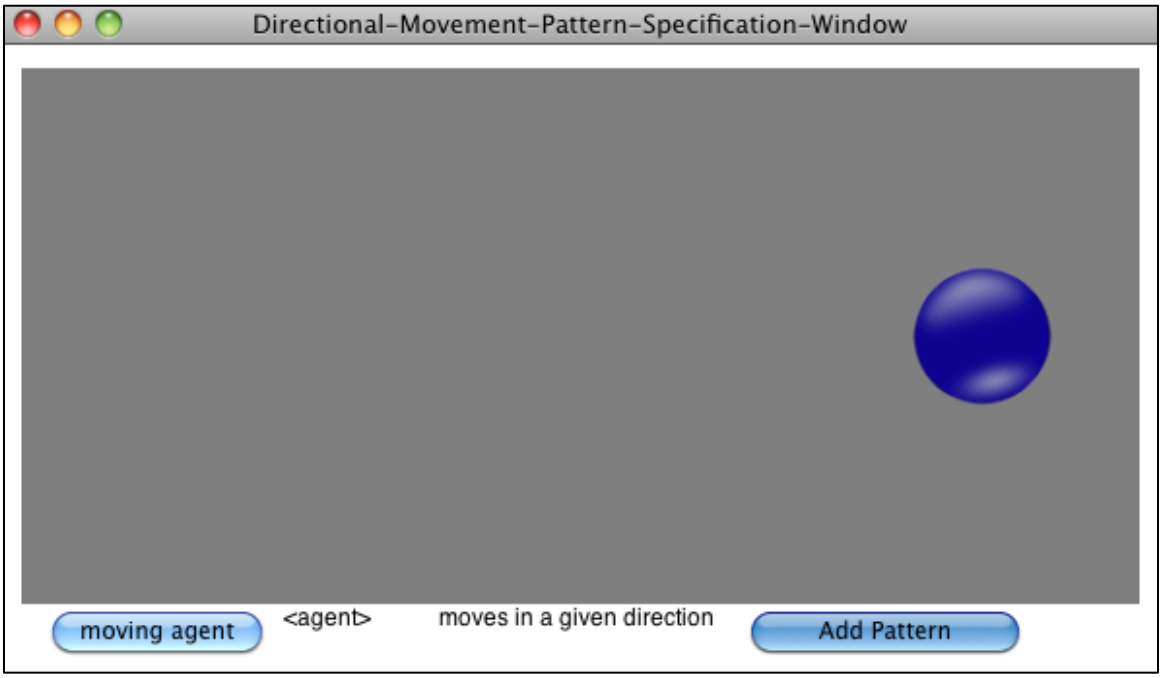


Figure 140: Directional Movement Pattern Window

The Directional Movement Pattern Window enables a user to select a "moving agent" that will move in a given direction at a constant speed.

The following picture depicts three sequential frames, from left to right, of the Directional Movement Pattern interacticon.

Figure 141: Three Sequential Frames Of The Directional Movement Pattern Interacticon

The blue disk in the Directional Movement Pattern interacticon moves from left to right at a constant velocity.

B.13 The Generate Pattern Window

The following picture depicts the Generate Pattern Window.

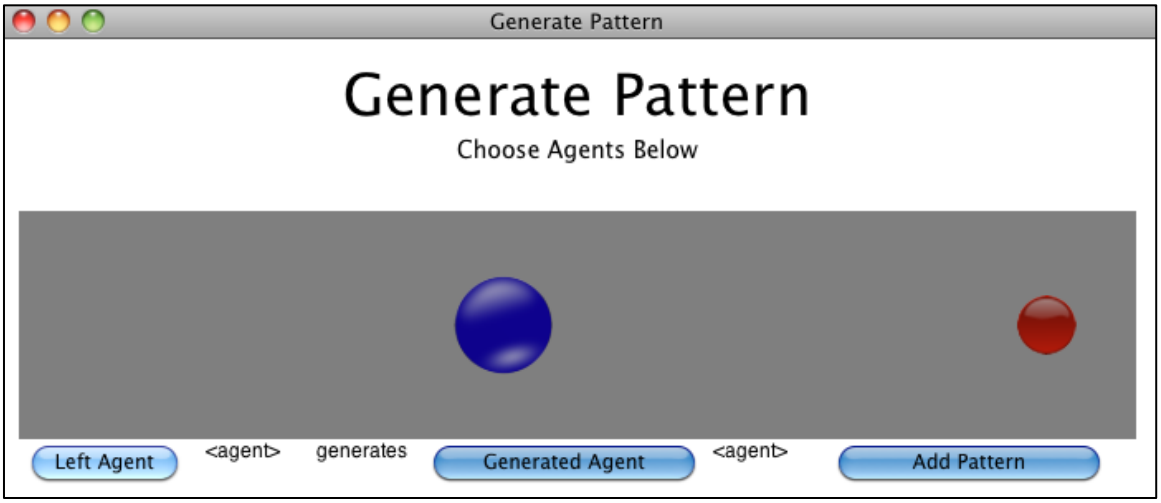


Figure 142: The Generate Pattern Window

The Generate Pattern Window allows a user to specify a Generator Agent by using the "Left Agent" button and an agent to be generated by using the "Generated Agent" button.

The following picture depicts three sequential frames of the Generate Pattern interacticon.

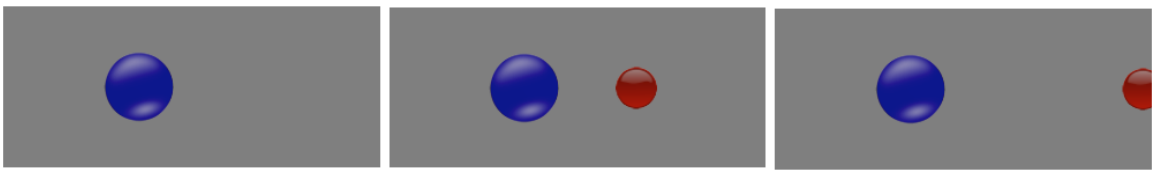


Figure 143: Three Sequential Frames Of The Generate Pattern Interacticon

In the Generate Pattern interacticon, the small red disk appears next to the large blue disk and starts moving away from the blue disk at a constant velocity.

B.14 The Data Pattern Window

The following picture depicts the Data Pattern Window.

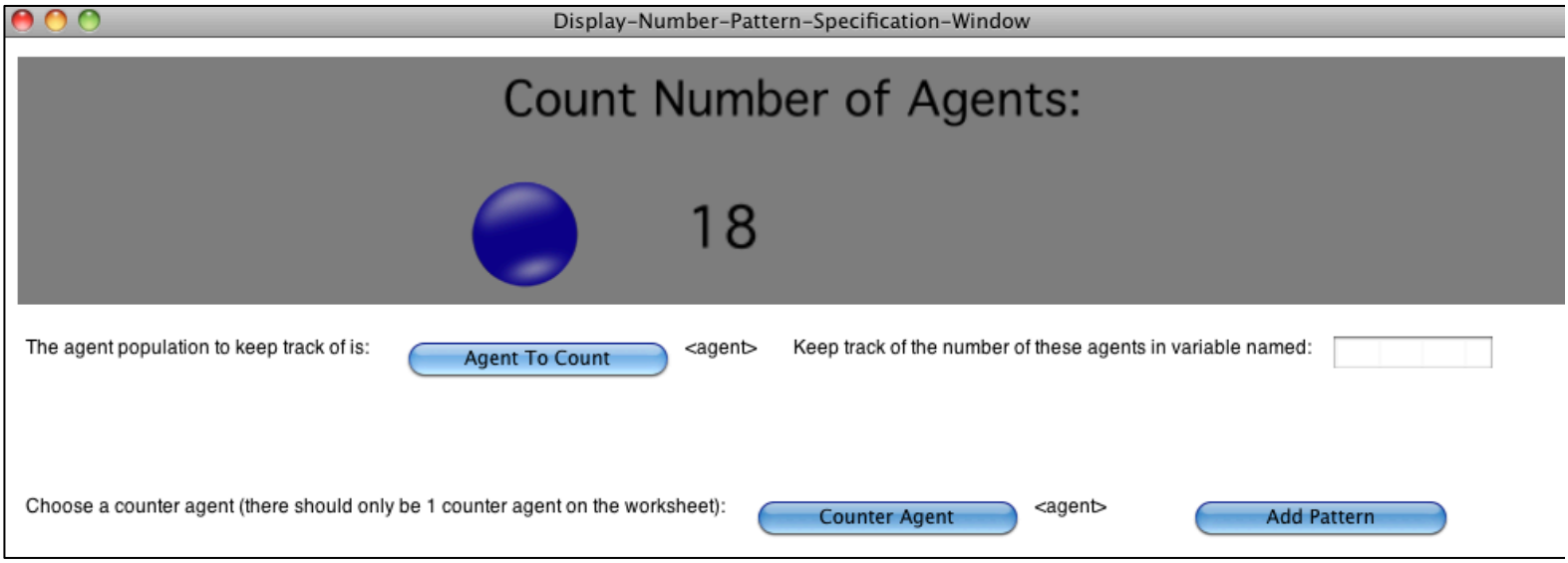


Figure 144: The Data Pattern Window

The Data Pattern Window allows the user to specify an "Agent To Count." On the right side of the window is a text box that enables a user to type in a variable to store this agent population data. Finally the "Counter Agent" button enables a user to specify the agent that does the counting. The interacticon for the data pattern consists of a number that slowly increments with time and a stationary blue disk representing the agent being counted.

APPENDIX C: The Predator/Prey Unit Tutorial/Worksheet

The tutorial/worksheet for the Predator/Prey unit is split up into four days plus an additional experiments section that can be completed after the four days of worksheets. Many students worked ahead of the actual days listed on the worksheet.

C.1 Predator/Prey Worksheet Day 1

WORKSHEET AND VISUAL CHEAT SHEET FOR
PREDATOR PREY SIMULATION: **DAY 1 (Return After Class!)**

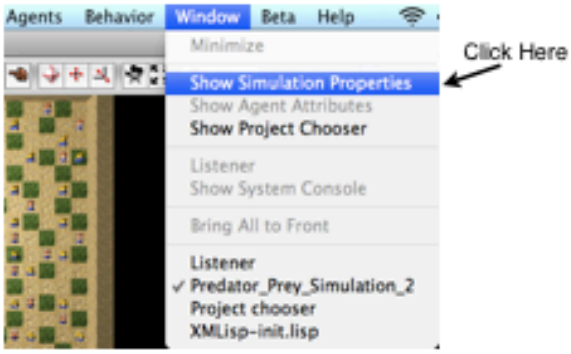
NAME: _____

GRADE: _____ TEACHER: _____

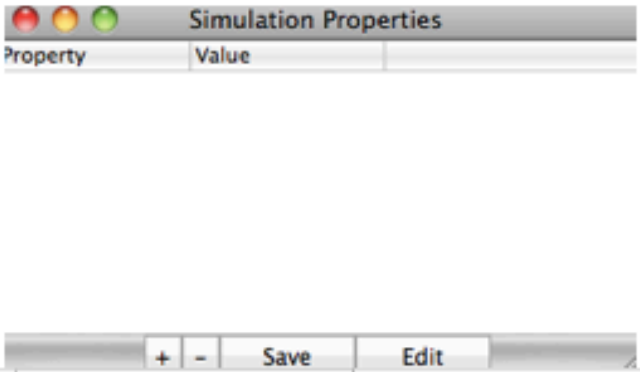
PERIOD:_____ COMPUTER NUMBER:_____

Follow along in class as we show you how to find The Simulation You Will Use for Today

1. Open the Simulation Properties Window



2. You Should See The Following Window



2. Hit Play In The Main Window To Start The Simulation Make Sure You Can Still See The Simulation Properties Window

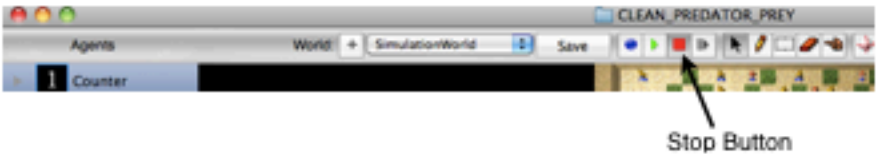
3. You Should See A Timer And 3 Variables In The Simulation Properties Window. These Refer to Different Populations In The Simulation. These Are Our Initial Populations.

What is our initial Fox Population? _____

What is our initial Rabbit Population? _____

What is our initial Grass Population? _____

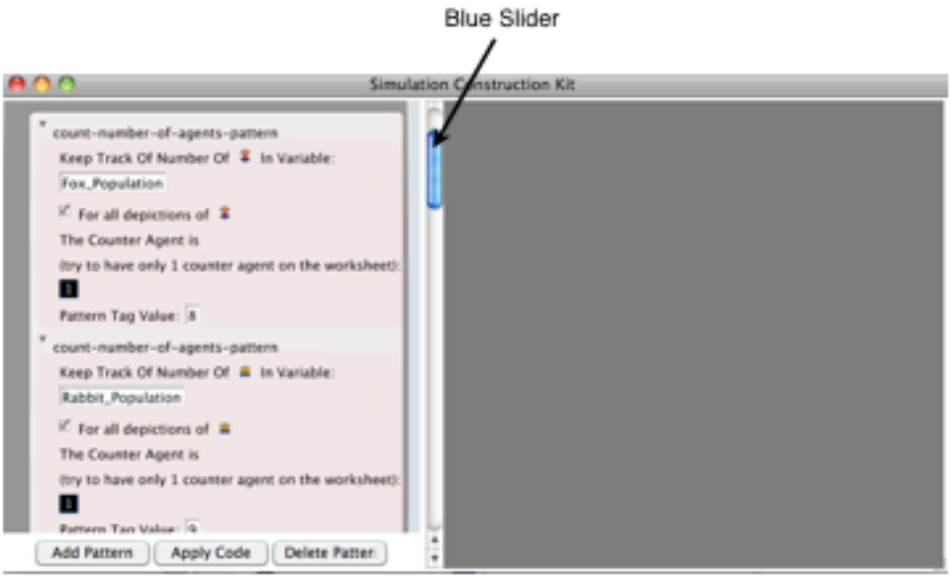
4. Hit Stop On The Simulation



Pretty Boring. Now Let's MAKE OUR AGENTS MOVE!

5. Hit The Blue Circle Button. This Button Allows Us To Add A Pattern

You Should See This Window. This Window shows all the three patterns already implemented. Play with the blue slider to scroll down and see the patterns.



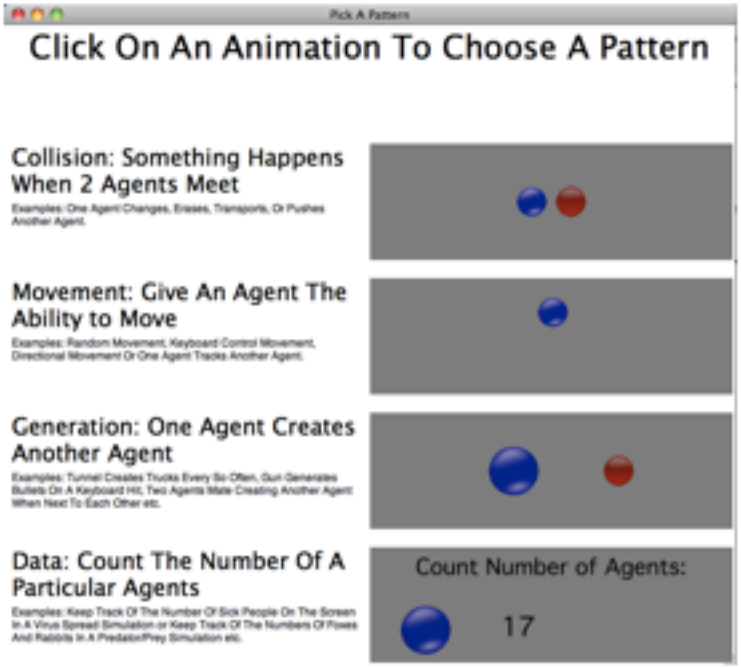
What Do You think these patterns do (it's alright if you don't know)? _____

_____

We are now going to make our Fox and Rabbit Move Randomly.

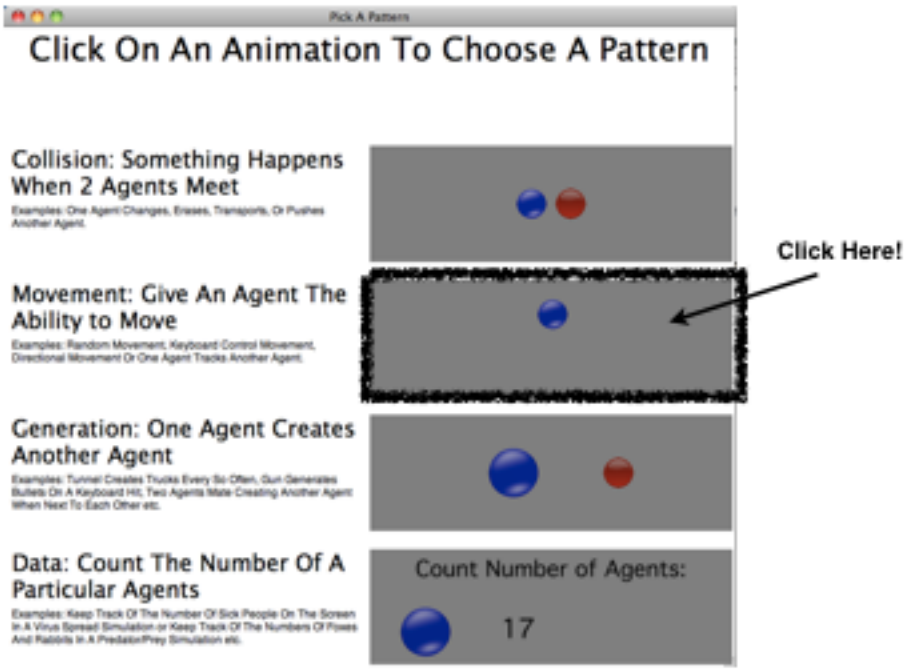
6. Hit The "Add Pattern" Button Located at the Bottom Left

You Should See The Following Window With Animations

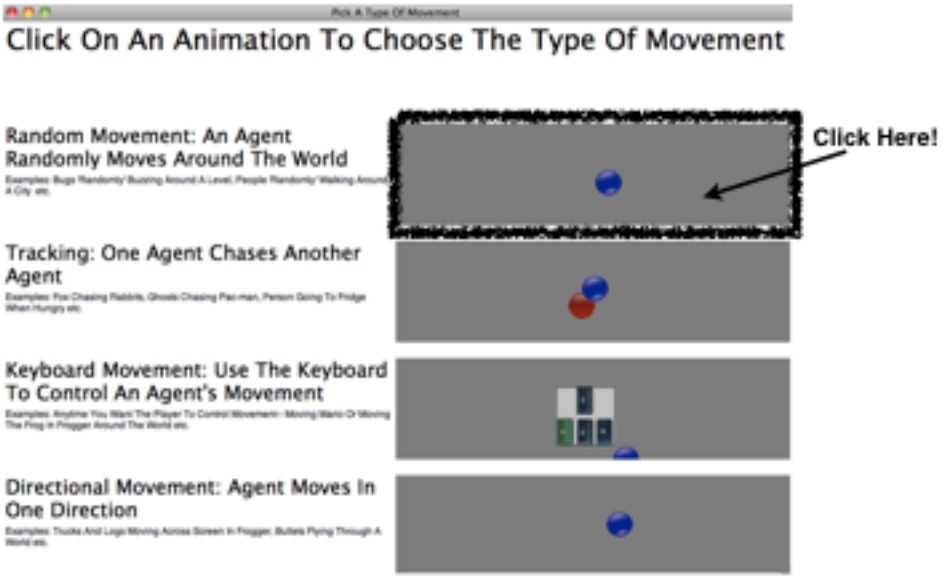


Don't Worry About All The Words Right Now. They Just Describe The Types Of Patterns You Can Create.

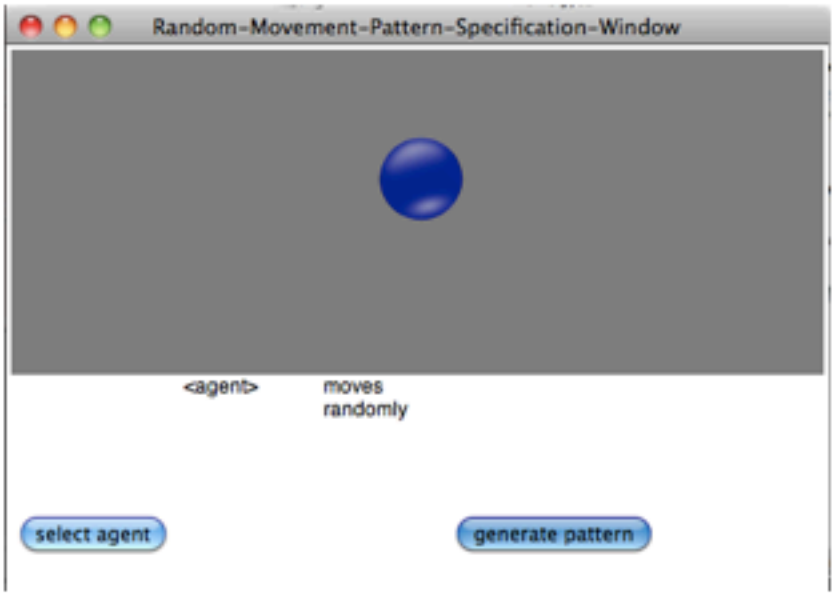
7. We Want To Make The Fox Move Randomly So Select The Movement Pattern By Clicking on the Movement Animation



8. You Should See The Following Window That Shows You All The Movement Animations. Click On The Random Movement Animation



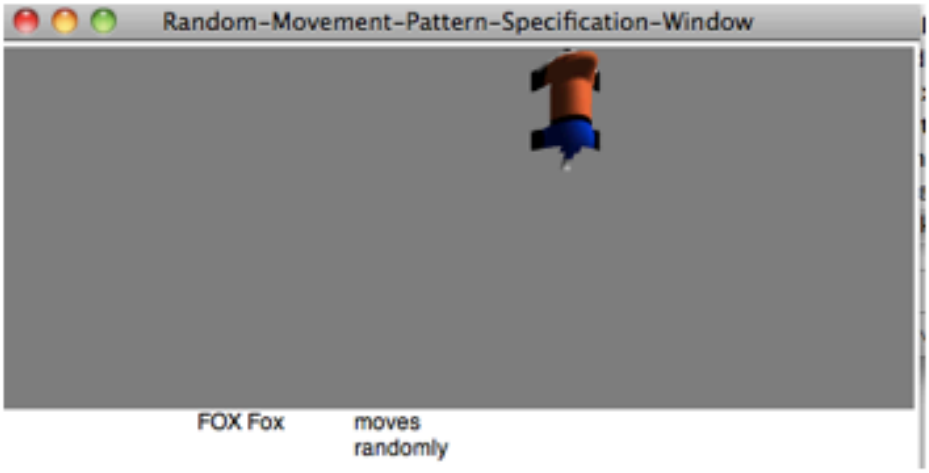
You should now see the following window. This is where we will add our pattern.



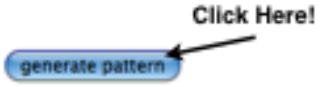
9. Click On The "Select Agent" Button At The Bottom Left

10. This Will Pop Up An Agent Selection Box. Select The Middle Fox As Shown Here

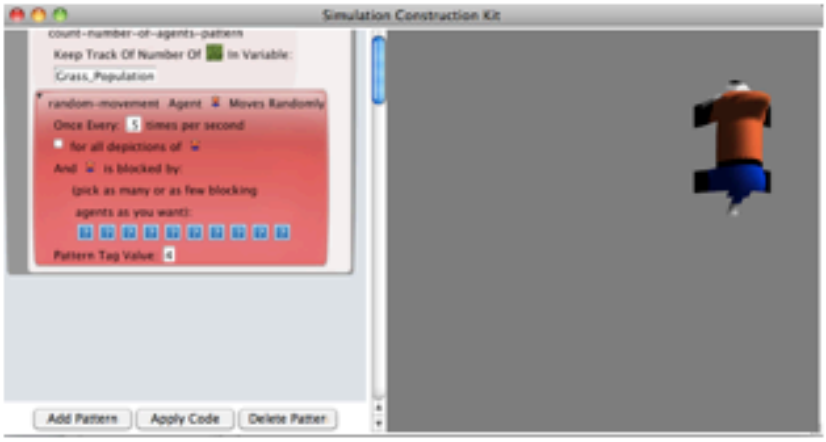
You Should see the Fox Moving around your screen



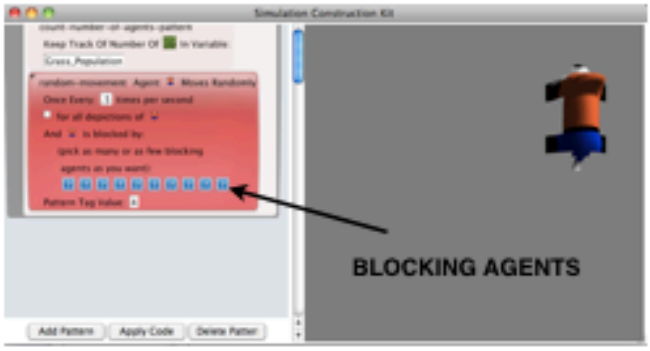
11. Click the "Generate Pattern" Button At The Bottom Right



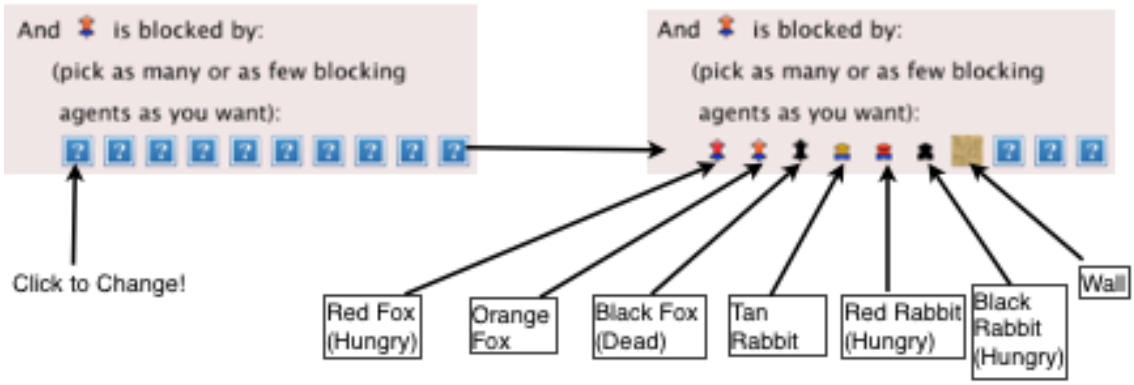
You Should See Your Pattern And Animation In the Following Window. If You Read The Pattern You Implemented It Says "Agent Fox Moves Randomly Once Every .5 Times Per Second"



12. The Pattern Gives Us A Place To Add Agents That Block The Fox's Movement At the Bottom.



Let's Make The Fox's Movement Be Blocked By All 3 Foxes, All 3 Rabbits and the Wall Agent. This Will Make The Simulation Look Cleaner Later.



Now Let's Watch Our Fox Move!

13. Select the "It's A New World" Worksheet From the World Drop Down Menu

You Should See A Small Rabbit With 1 Fox and 1 Rabbit. This Level Will Allow Us To Test Agent Behaviors and See Them More Easily

14. Click Play And The Fox Should Move Randomly (but not move on the wall or the Rabbit)

Hit The "Stop Button" When You're Done Seeing The Fox Move

Great Job! If We Have Time We Will Hand Out Worksheet 2; Otherwise Tomorrow We Will Make The Rabbit Move, And Then Make The Rabbit And Fox Get Hungry!

C.2 Predator/Prey Worksheet Day 2

WORKSHEET AND VISUAL CHEAT SHEET FOR PREDATOR PREY SIMULATION: DAY 2 (Return After Class!)

NAME: _____

GRADE: _____ TEACHER: _____

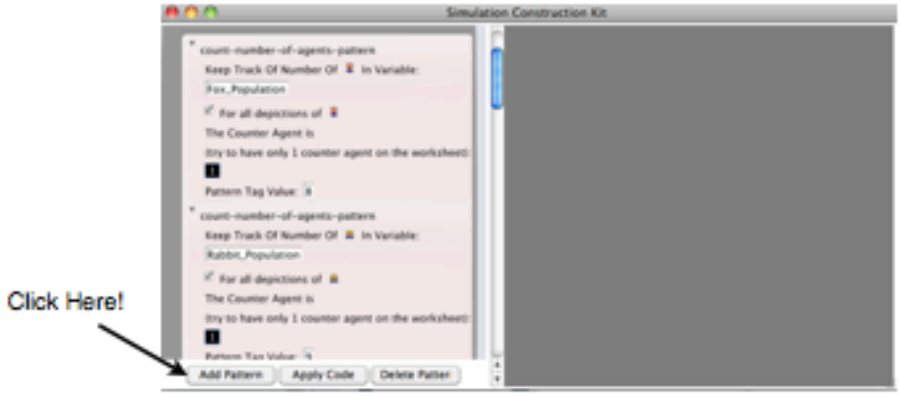
PERIOD:_____ COMPUTER NUMBER:_____

Follow along in class as we show you how to find your simulation you did yesterday and open it.

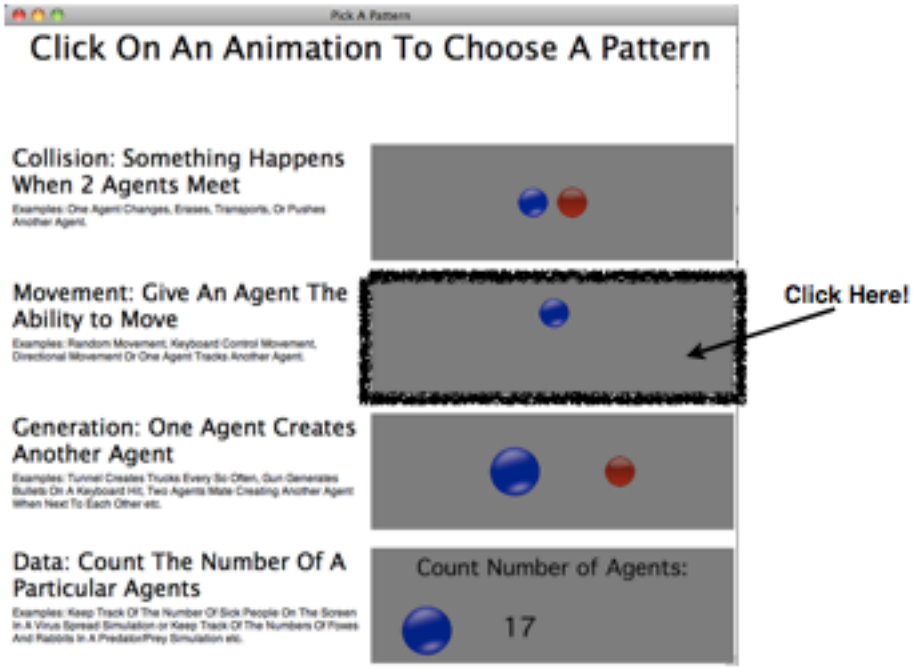
We Will Now Make The Rabbit Move Randomly. This Is Very Similar To How We Made The Fox Move Randomly

1. Hit The Blue Circle Button. This Button Allows Us To Add A Pattern

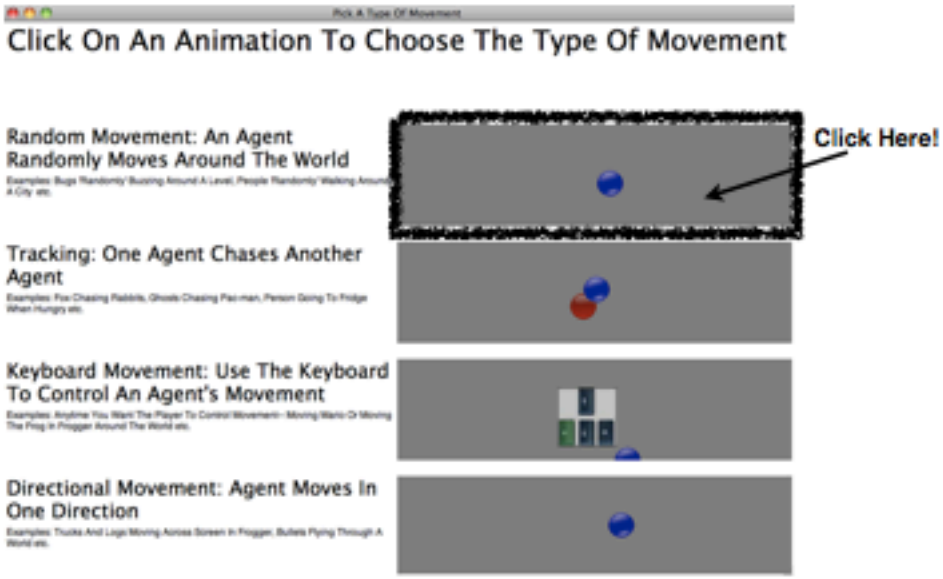
2. Hit The "Add Pattern" Button Located at the Bottom Left



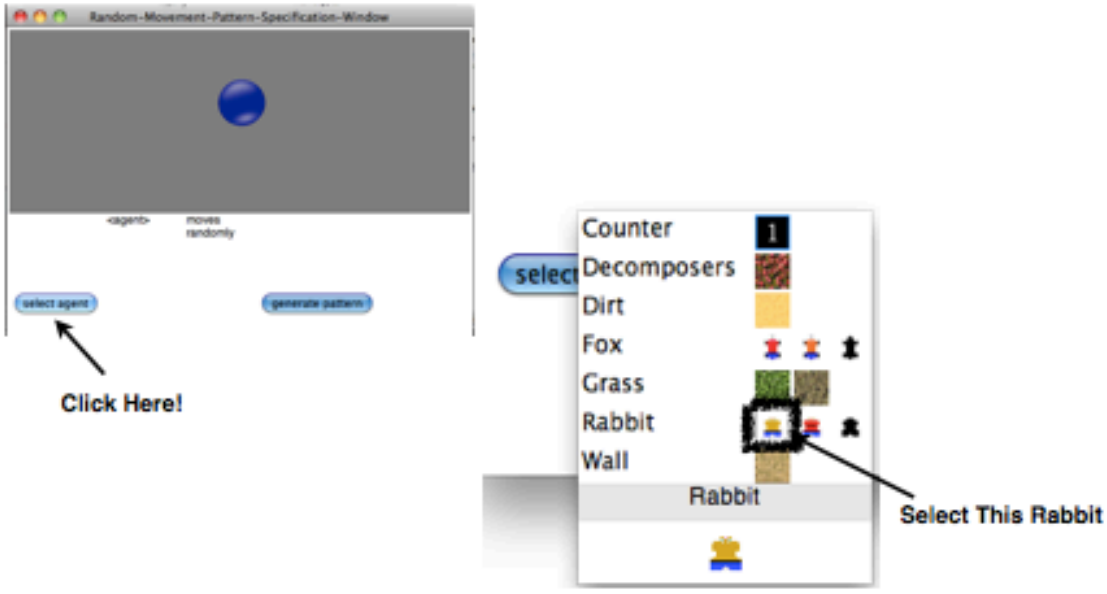
3. Click On The Movement Animation



4. This Will Bring Up The Movement Window. Click On The Random Movement Animation



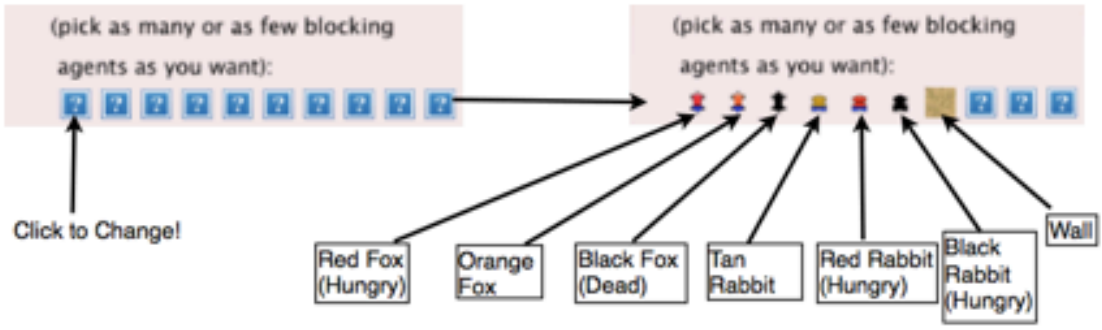
5. Click on The "Select Agent" Button At The Bottom And Select The Rabbit On the Left.



6. Click The "Generate Pattern" Button In The Bottom Right

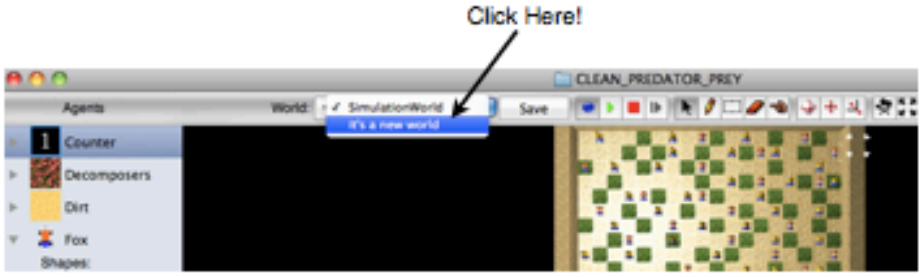
You Should See The Following

7. Like We Did With The Fox Let's Make Sure The Rabbit Is Blocked By All 3 Foxes, All 3 Rabbits and The Wall

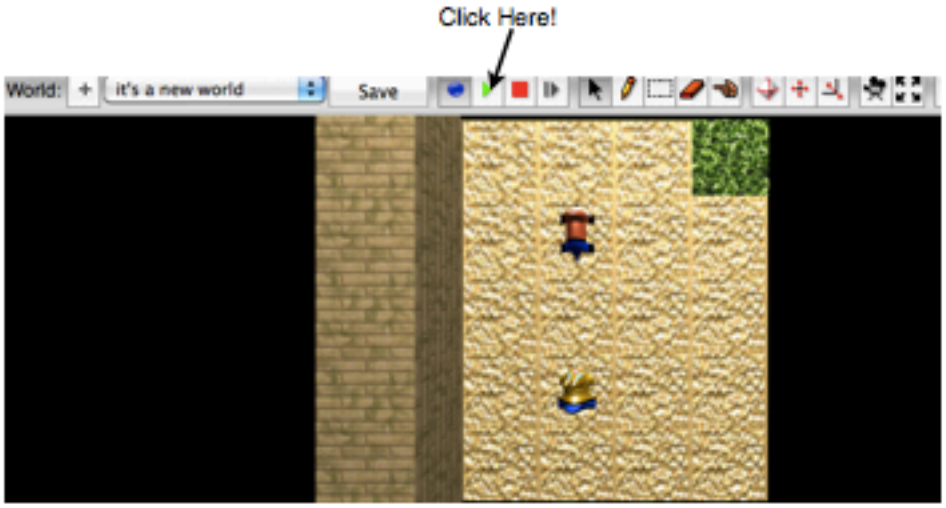


Now Let's Watch Rabbit and Fox Move!

8. If It's Not Already Selected, Select the "It's A New World" Worksheet From the World Drop Down Menu



9. Click "Play" And The Fox And Rabbit Should Move Randomly



10. Hit The "Stop Button" When You're Done

Give A Reason Why This Movement Might Be Unrealistic?

_____

_____

Now Let's Make Our Agents Get Hungry Over Time

We Are Going to Have the Dirt Change the Regular Fox And Regular Rabbit Into a Hungry Fox And Hungry Rabbit With a Given Percent Chance

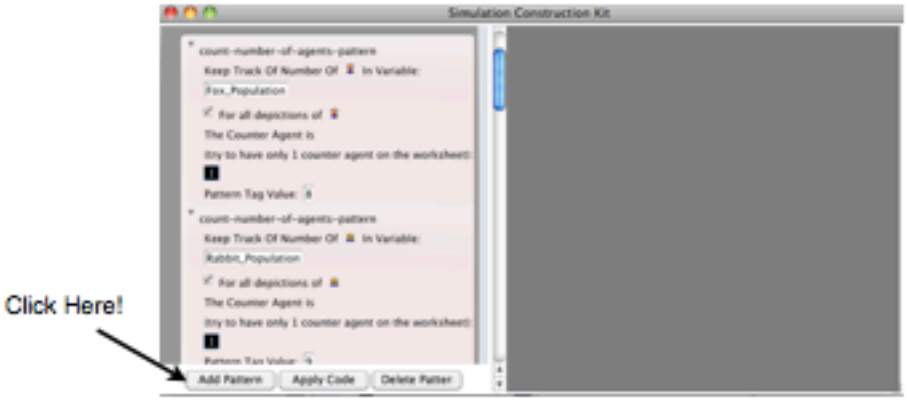
What Pattern Do You Think We Will Use To Do This?

_____

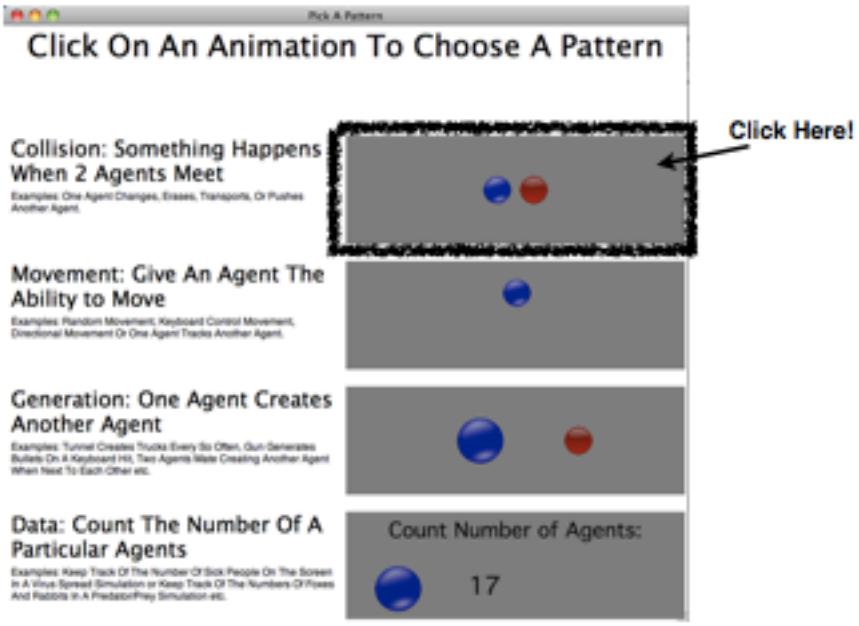
_____

Click on the Blue Circle Button at the top of the screen

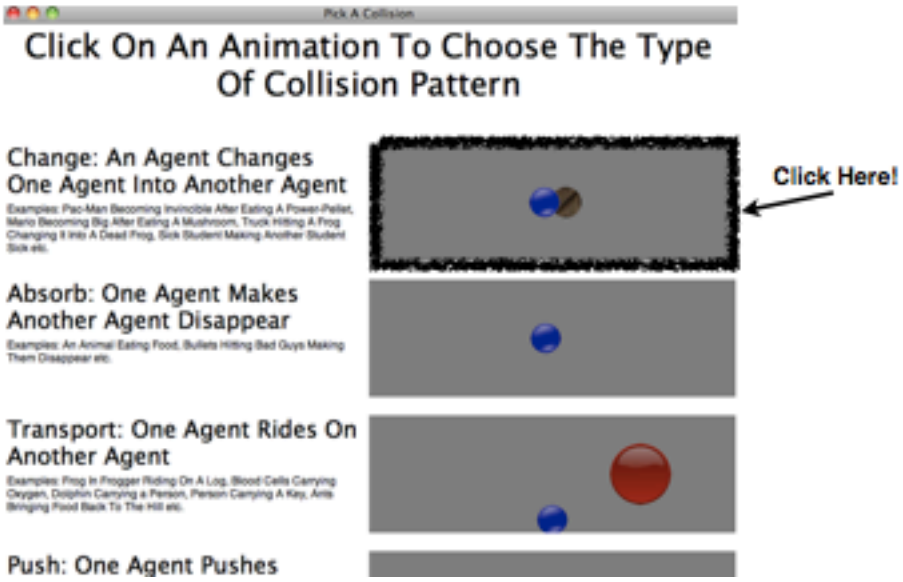
11. Hit The "Add Pattern" Button Located at the Bottom Left



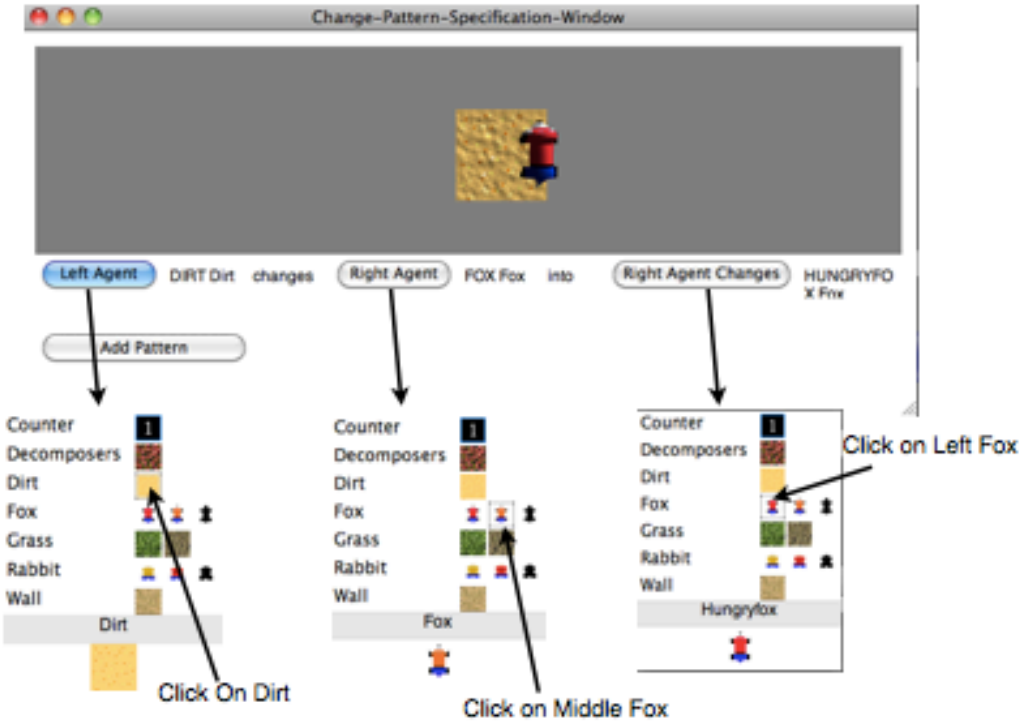
12. Now We Are Going To Click On The COLLISION Animation



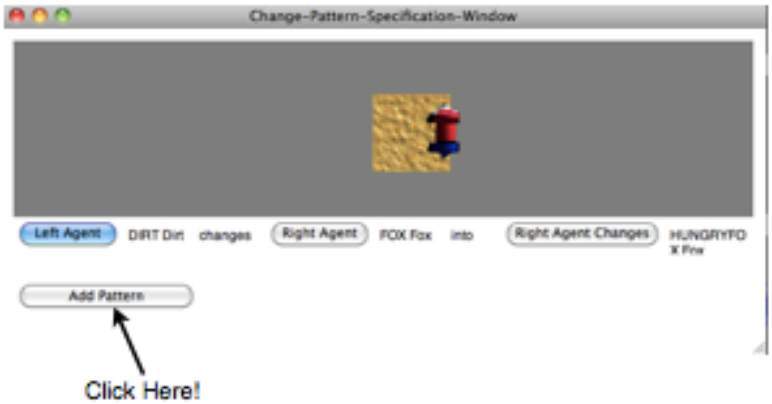
13. On The Next Window we are going to click on the CHANGE animation



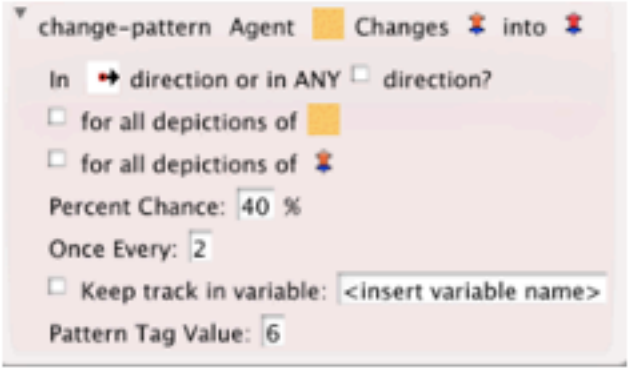
14. We Are Going to Have the Dirt Change the Regular Fox Into a Hungry Fox. The Following Picture Shows You How To Do This



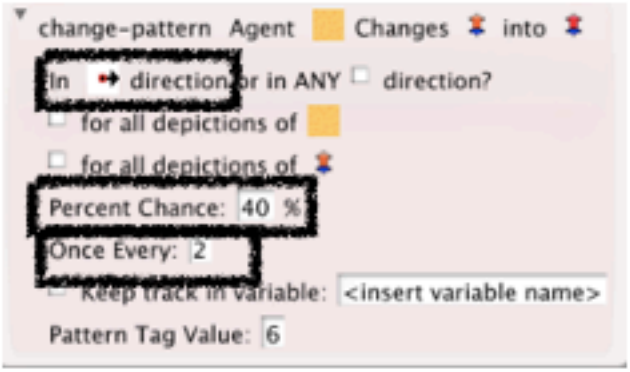
15. When you are done click the "Add Pattern" button at the bottom left



At This Point You Should See Something Close The Following (not exactly though)

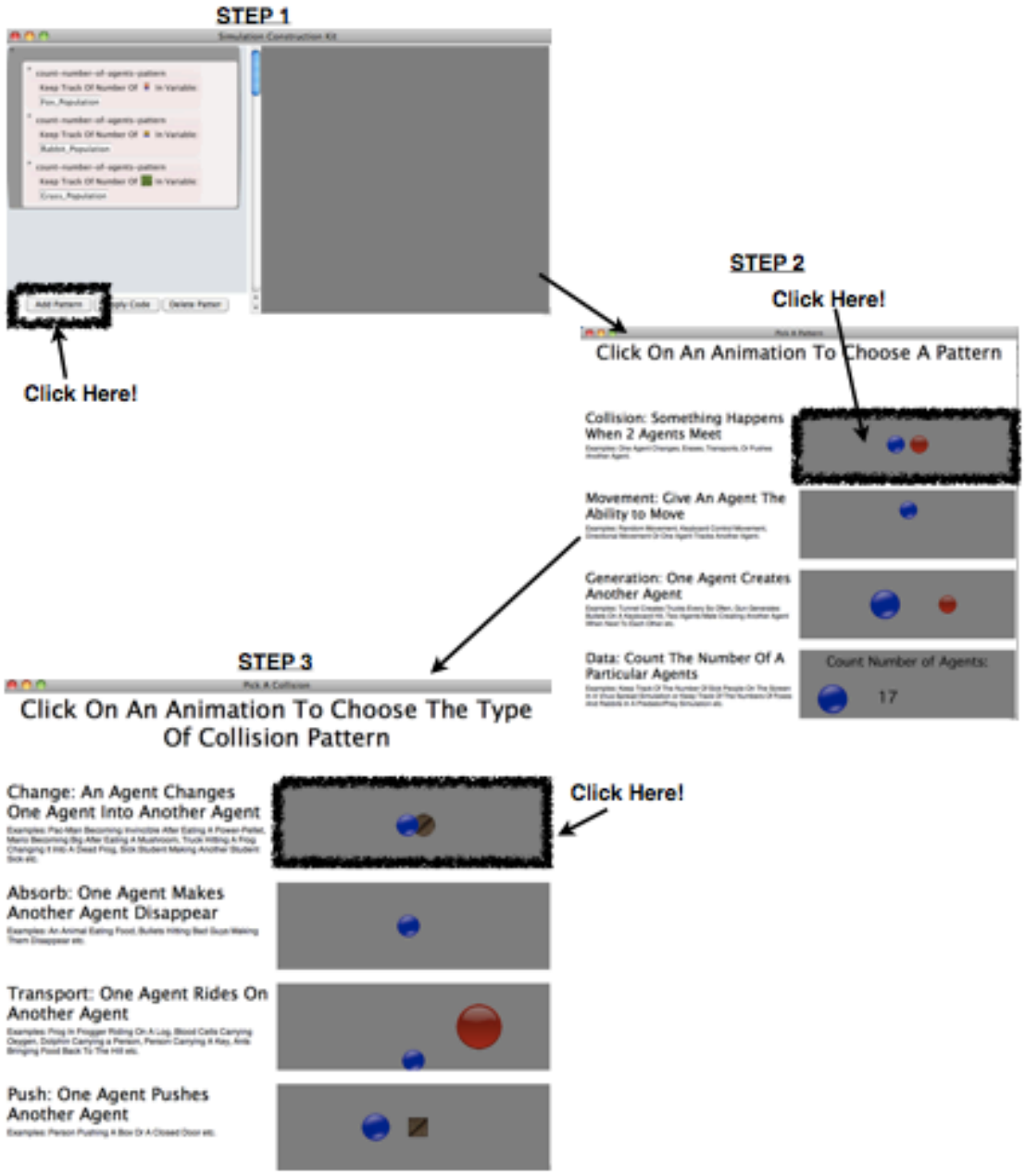


16. We Are Going To Make The Dirt Change The Fox Into A Hungry Fox In The --> Direction With a 40% Chance Once Every 2 Seconds. Make The following 3 Changes

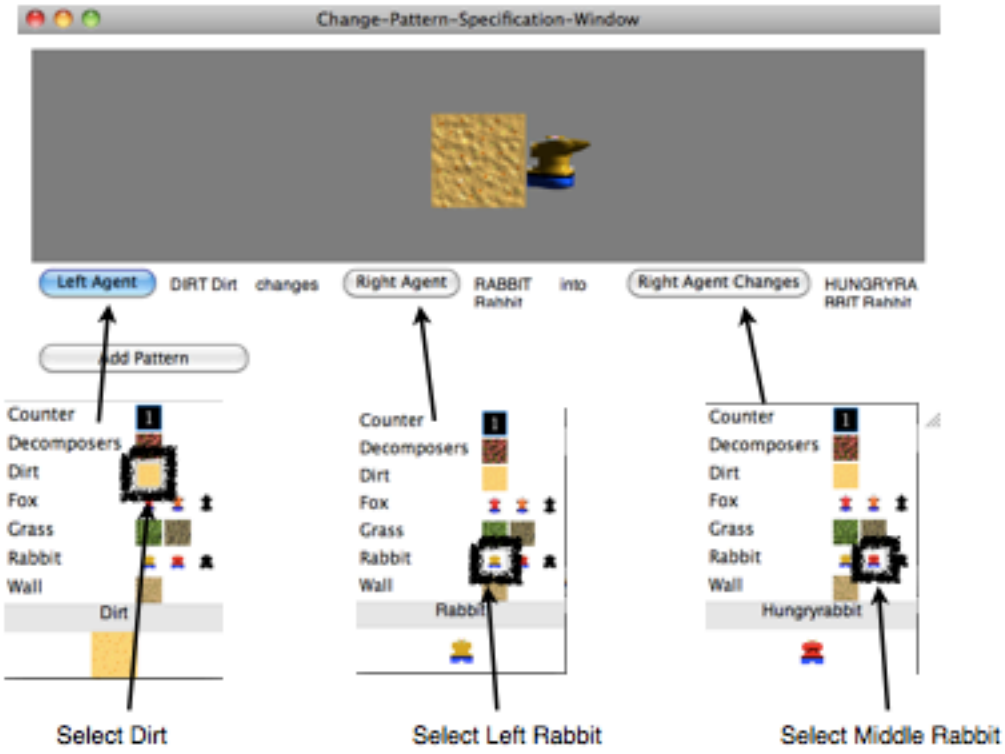


Now Let's Do The Same For the Rabbit. The Steps On The Next Page Will Explain How To Do This

17. We Will Go Through The Same Steps But This Time We Will Make The Ground Turn The Regular Rabbit Into A Hungry Rabbit. The Following Pictures Outline These Steps In Order

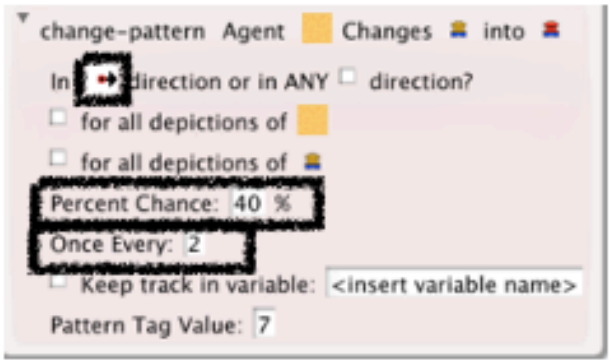


18. Now We Will Make The Dirt Change The Rabbit Into A Hungry Rabbit As Follows

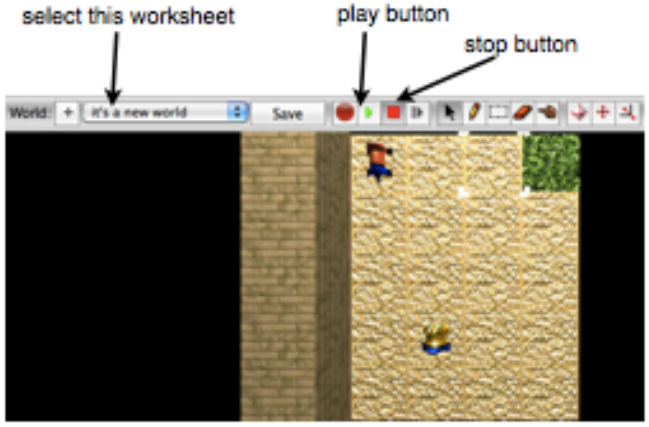


19. When you are done click the "Add Pattern" button at the bottom left

20. Like We Did With The Fox, Make the Dirt Change the Rabbit into a Hungry Rabbit in the *Right Direction* with a *40% chance* once *Every 2 Seconds*



GREAT! Now Both Our Agents Should Move And Get Hungry (Turn Red) Over Time. We Can Test This Out By Hitting Play On The "it's a new world" Worksheet. Note That When Our Agents Turn Red They Will Stop Moving. Hit The Stop Button When You're Done.



The Fox And Rabbit Are Moving Once Every .5 seconds. What are they using up as they move?

_____

_____

_____

As Time Goes On In The Simulation Are the Bunny and Fox More or Less Likely to Become Hungry? Is This Realistic?

_____

_____

_____

Instead of Stop Moving, What Do You Think The Hungry Fox should Do? What Should The Hungry Rabbit Do?

_____

_____

_____

Great Job! If We Have Time We Will Hand Out Worksheet 3; Otherwise Tomorrow We Will Make The Hungry Rabbit And Fox Move And Die If They Do not Get Food!

C.3 Predator/Prey Worksheet Day 3

WORKSHEET AND VISUAL CHEAT SHEET FOR
PREDATOR PREY SIMULATION: **DAY 3 (Return After Class!)**

NAME: _____

GRADE: _____ TEACHER: _____

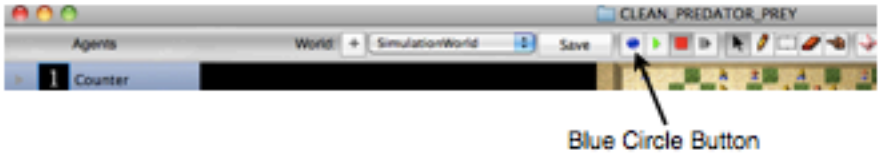
PERIOD:_____ COMPUTER NUMBER:_____

Follow along in class as we show you how to find your simulation you did yesterday and open it.

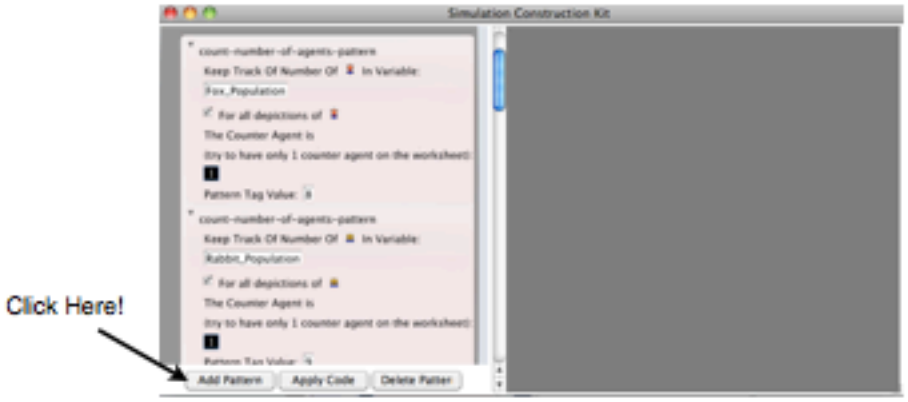
Today We Will Make The Fox And Rabbit Die If They Stay Hungry Long Enough And Decompose/Turn Into Grass. We will also have the Hungry Fox Chase and Eat The Rabbit and The Rabbit Chase And Eat The Grass

First, We Are Going To Have The Dirt Change The Hungry (Red) Rabbit and Hungry Fox into A Dead Rabbit And Fox.

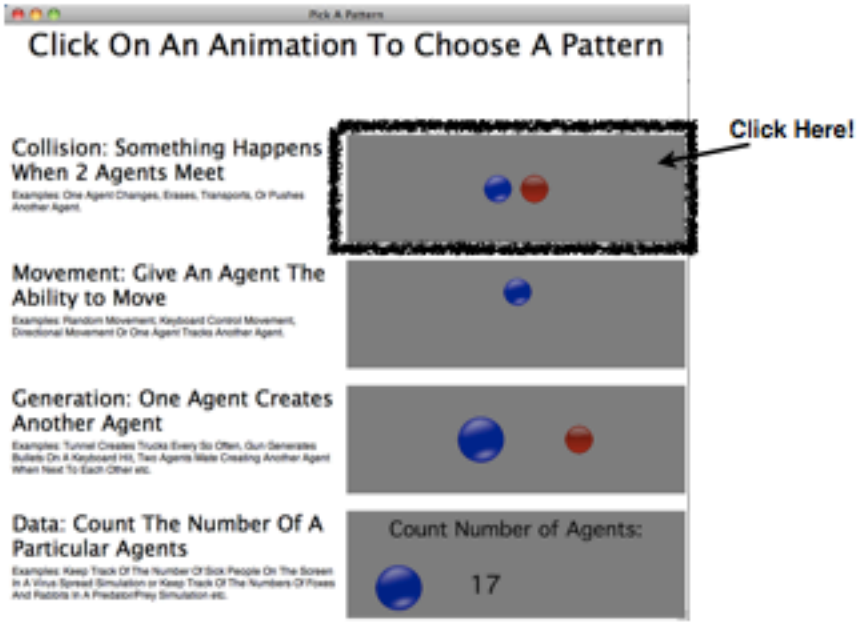
1. Hit The Blue Circle Button. This Button Allows Us To Add A Pattern



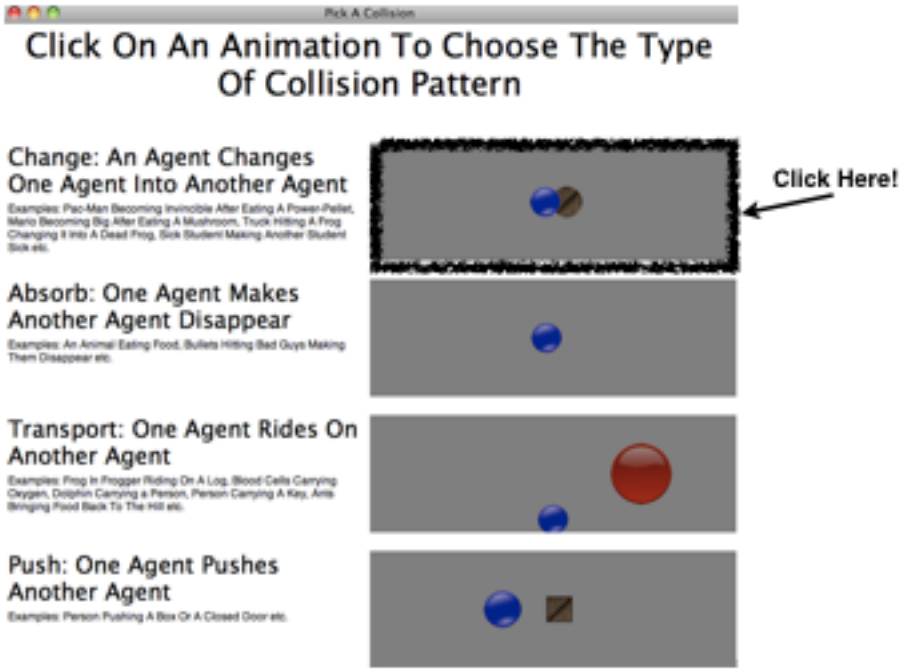
2. Hit The "Add Pattern" Button Located at the Bottom Left



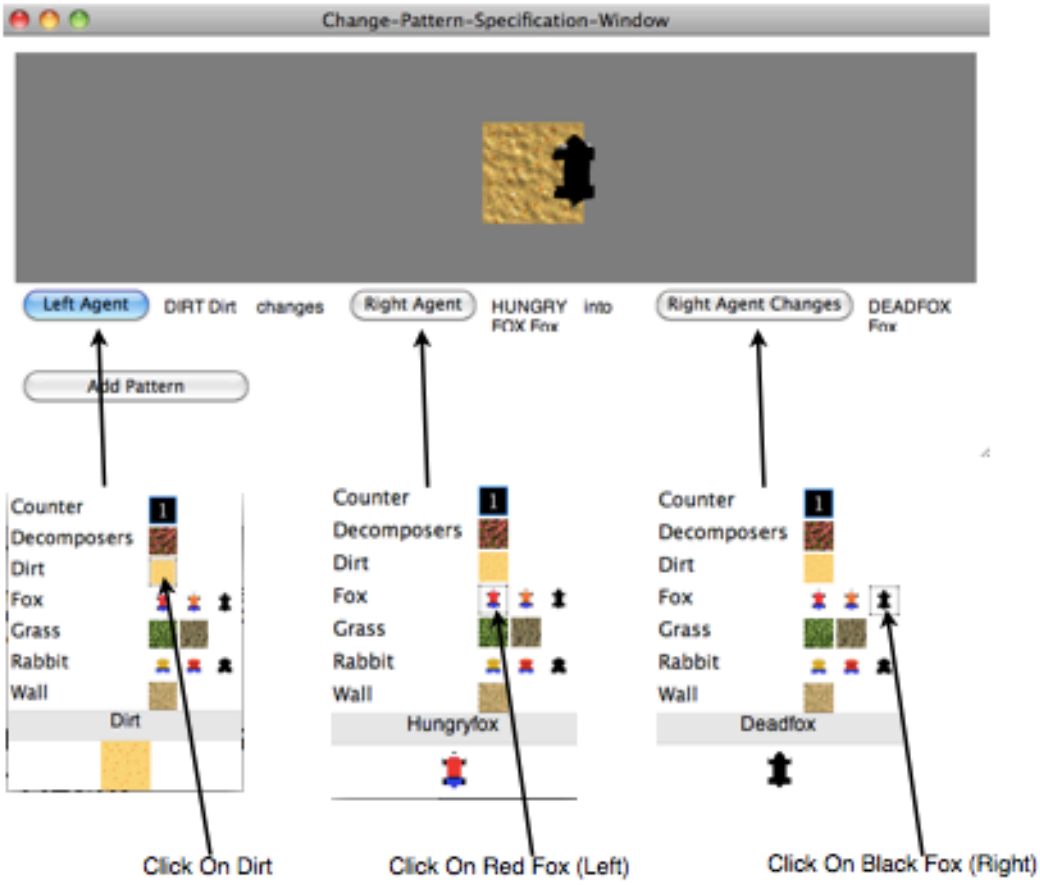
3. Now We Are Going To Click On The COLLISION Animation



4. On The Next Window we are going to click on the CHANGE animation



5. We Are Going to Have the Dirt Change the Hungry Fox Into a Dead Fox. The Following Picture Shows You How To Do This



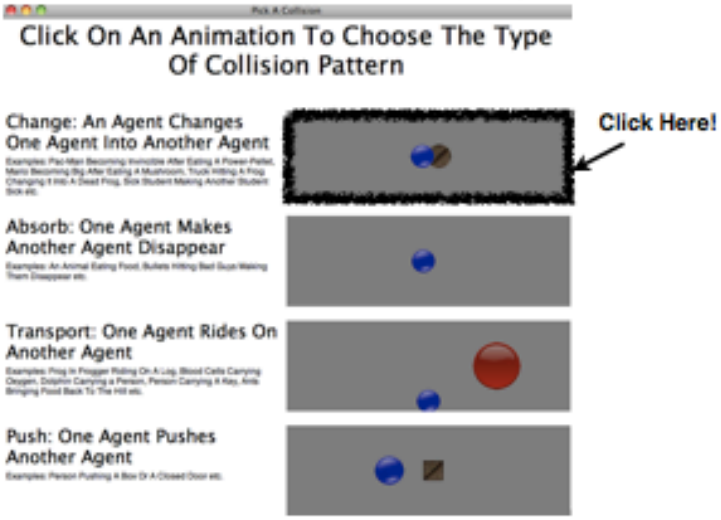
6. Make The Dirt Change The Hungry Fox To A Dead Fox Once Every 2 Seconds With a 40% Chance

7. We Will Go Through The Same Steps But This Time We Will Make The Ground Turn The Hungry Rabbit Into A Dead Rabbit. The Following Pictures Outline These Steps In Order

STEP 1

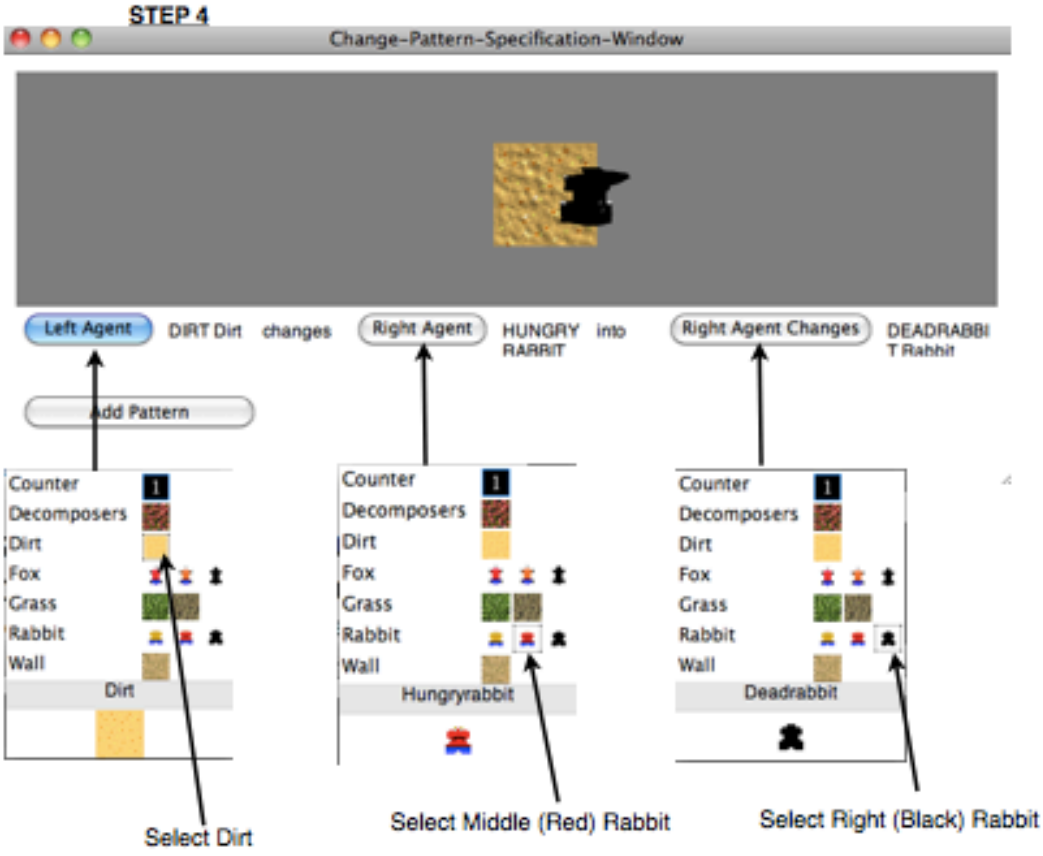
STEP 2

STEP 3



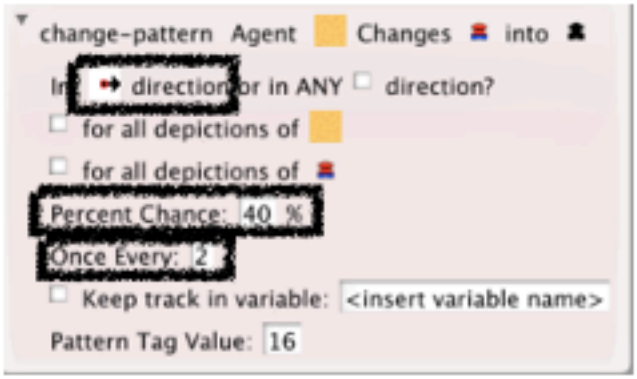
STEP 4 ON NEXT PAGE --->

STEP 4

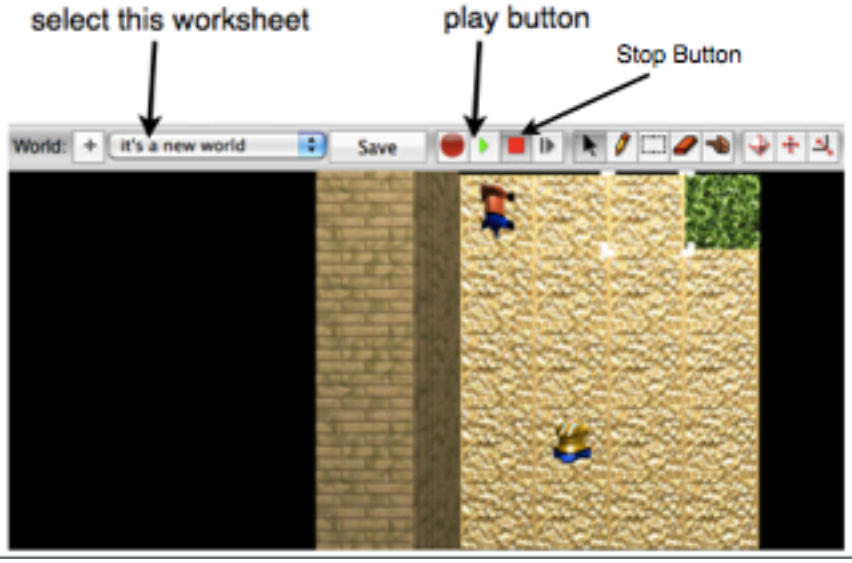


8. When you are done click the "Add Pattern" button at the bottom left

9. Like We Did With The Fox, Make The Dirt Change The Hungry Rabbit Into A Dead Rabbit In The *Right Direction* With A *40% Chance* Once *Every 2 Seconds*

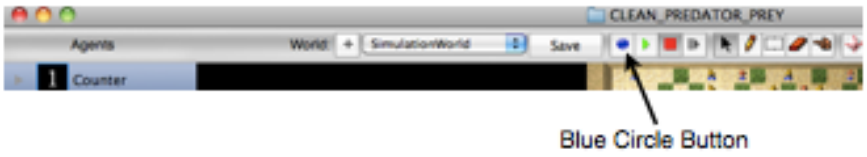


GREAT! Now Both Our Agents Should Move And Get Hungry (Turn Red) Over Time and Die (Turn Black). We Can Test This Out By Hitting Play On The "it's a new world" Worksheet. Note That When Our Agents Turn Red They Will Stop Moving Still Because We Haven't Programmed That Yet. Hit Stop When You're Done.

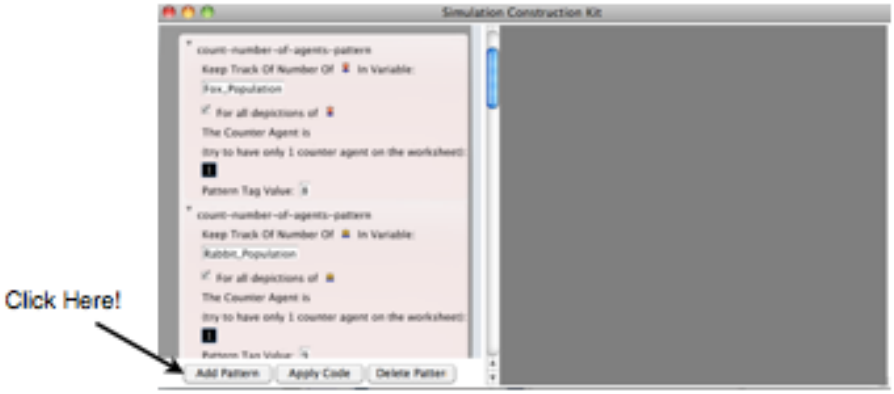


Now Let's Have the Dead Fox And Rabbit Decompose Into Grass

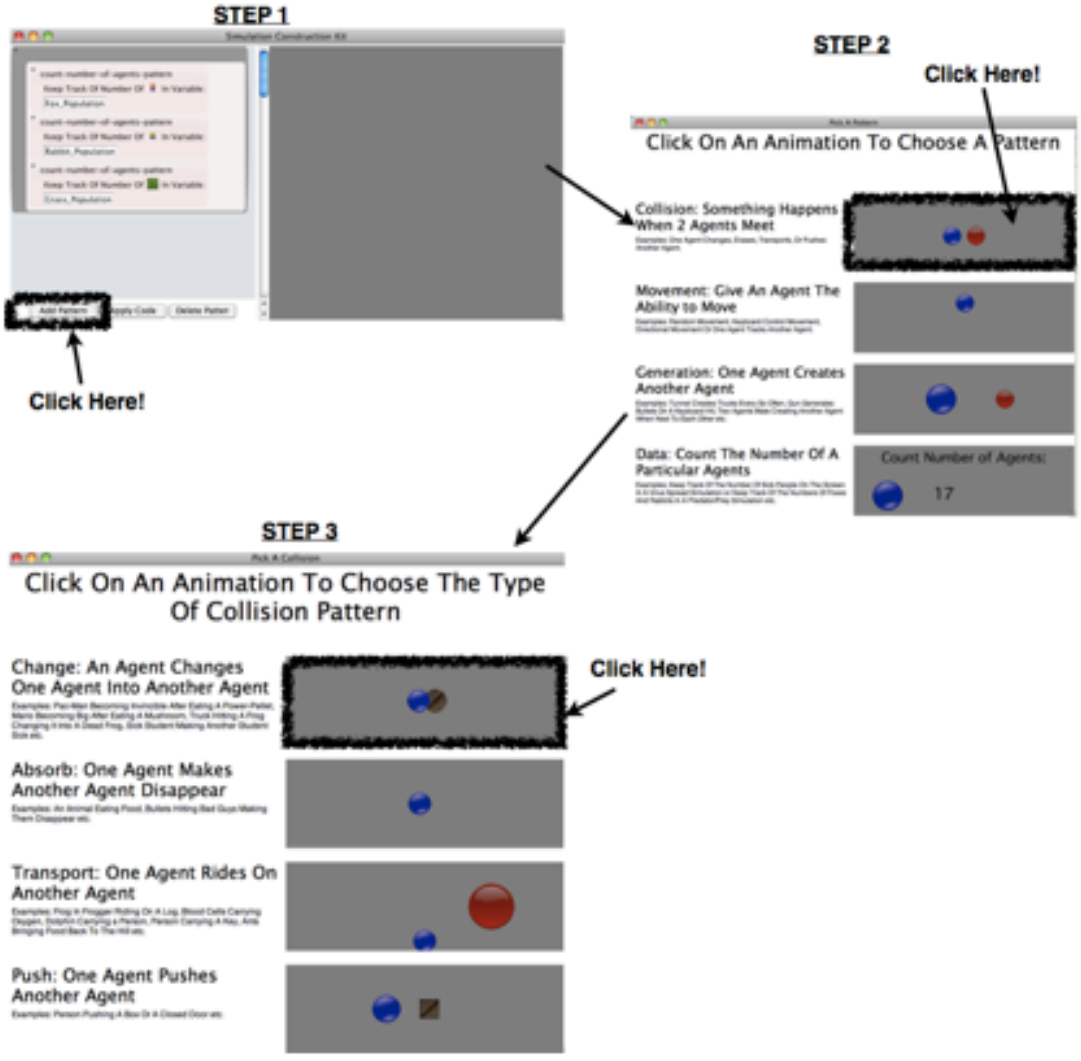
10. Hit The Blue Circle Button (If the Window In Step 11 Is Already Open You Don't Have To Do This)



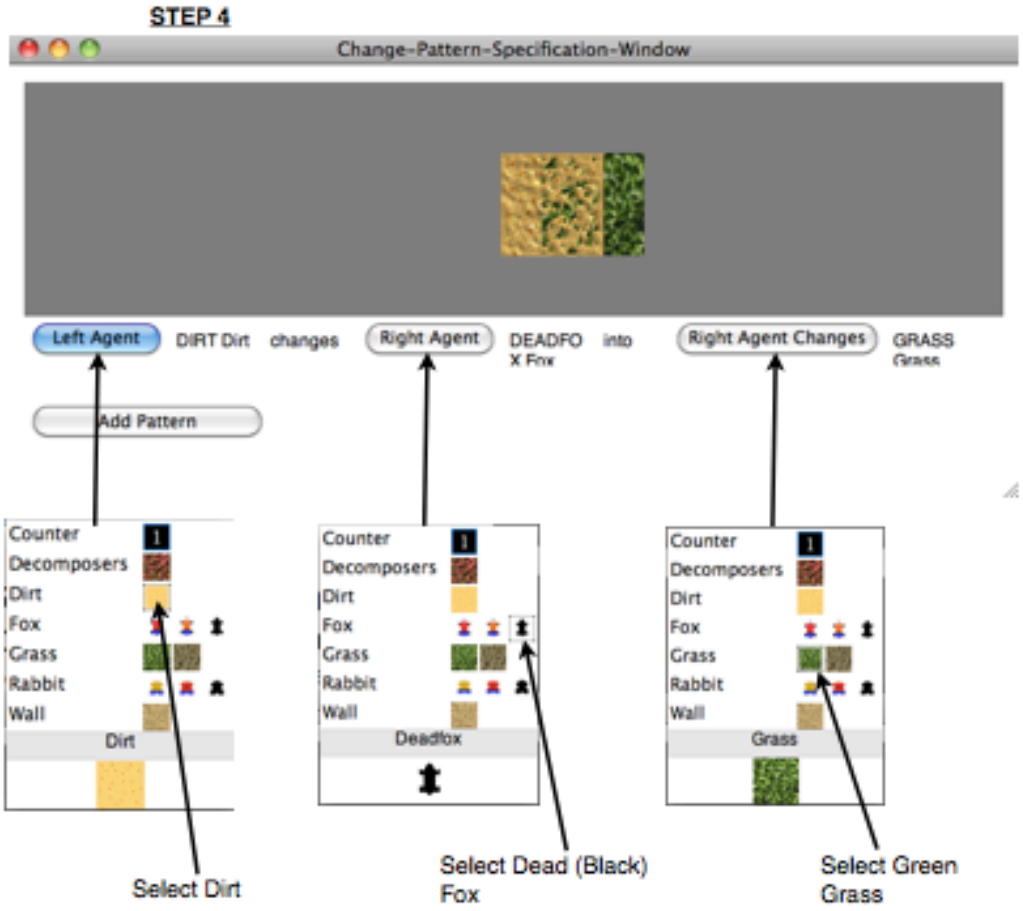
11. Hit The "Add Pattern" Button Located at the Bottom Left



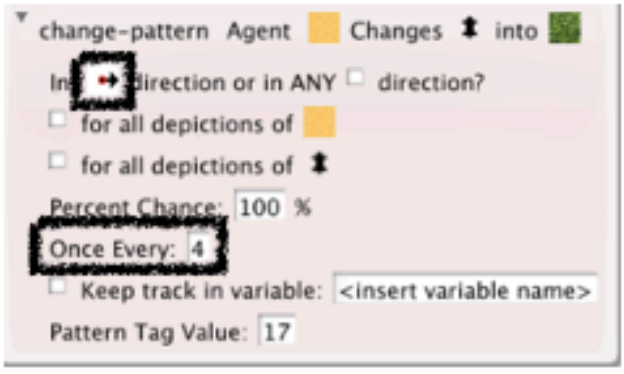
12. Follow The Steps Outlined Below.



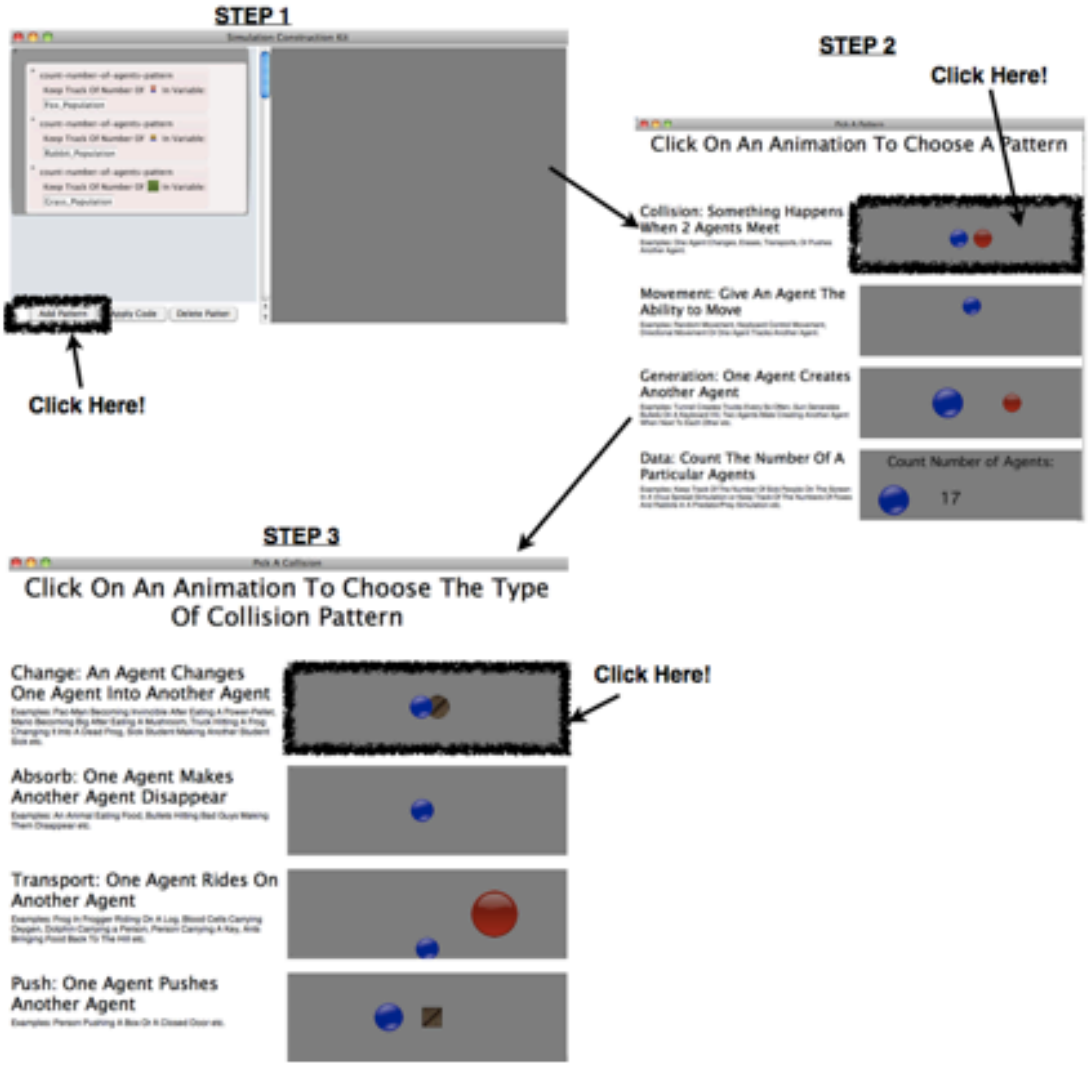
STEP 4 ON NEXT PAGE --->



13. We Want The Dirt To Change (Decompose) The Dead Fox Into Grass To The Right Once Every 4 Seconds.

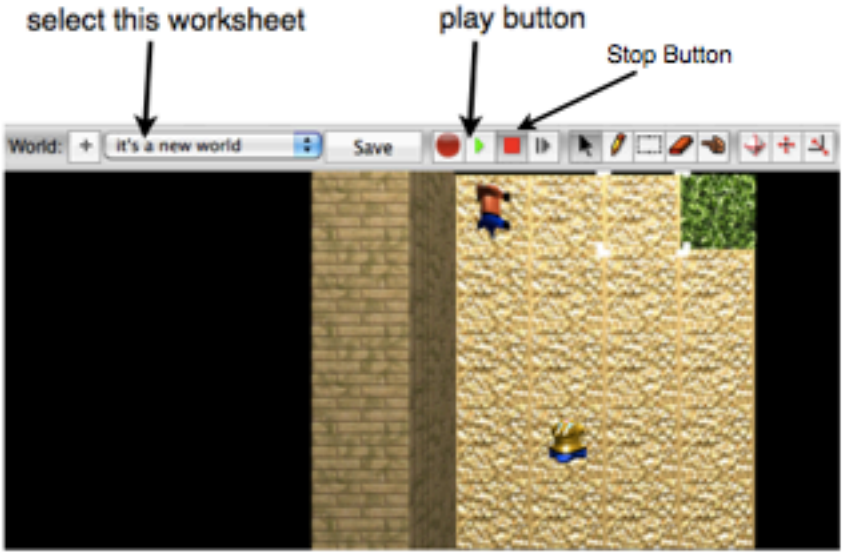


14. Now Let's Do The Same Thing For The Rabbit! Again Follow The Steps Outlined Below



STEP 4 ON NEXT PAGE --->

Awesome! Now Both Our Agents Should Move And Get Hungry (Turn Red) Over Time and Die (Turn Black), And Decompose Into Grass! We Can Test This Out By Hitting Play On The "it's a new world" Worksheet. Note That When Our Agents Turn Red They Will Stop Moving Still Because We Haven't Programmed That Yet. Hit Stop When You're Done



How Is The Fox And Rabbit Decomposing Unrealistic? (How Long Does It Take Them To Decompose vs. How Long Would It Actually take)?

_____

_____

_____

We Use The Dirt Agent To Make All Our Agent Changes. Why Do You Think We Use That Particular Agent? Is This Realistic (Do You Think Dirt Actually Does Any Of These Things In Real Life?).

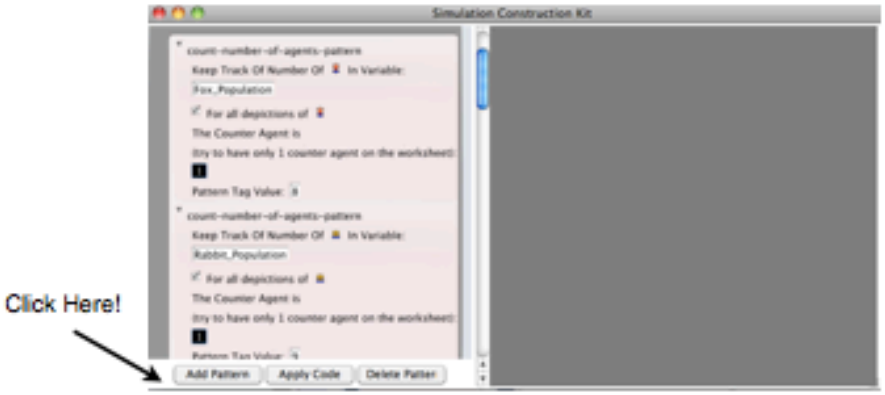
_____

_____

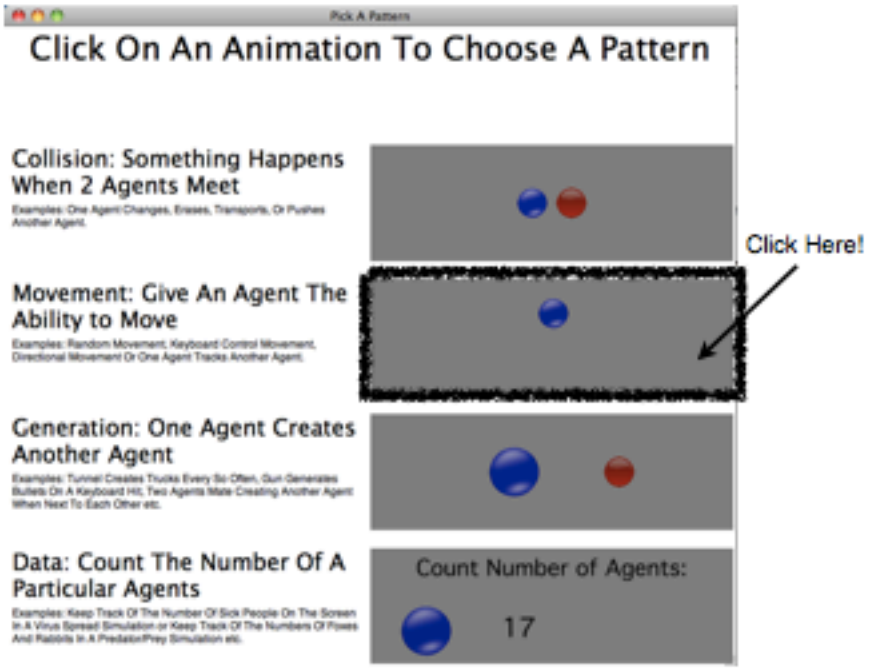
_____

Now We Are Going To Make The Hungry Fox Chase The Rabbit And The Hungry Rabbit Go After The Grass. To Do This We Are Going To Use The Tracking Pattern Found In Movement.

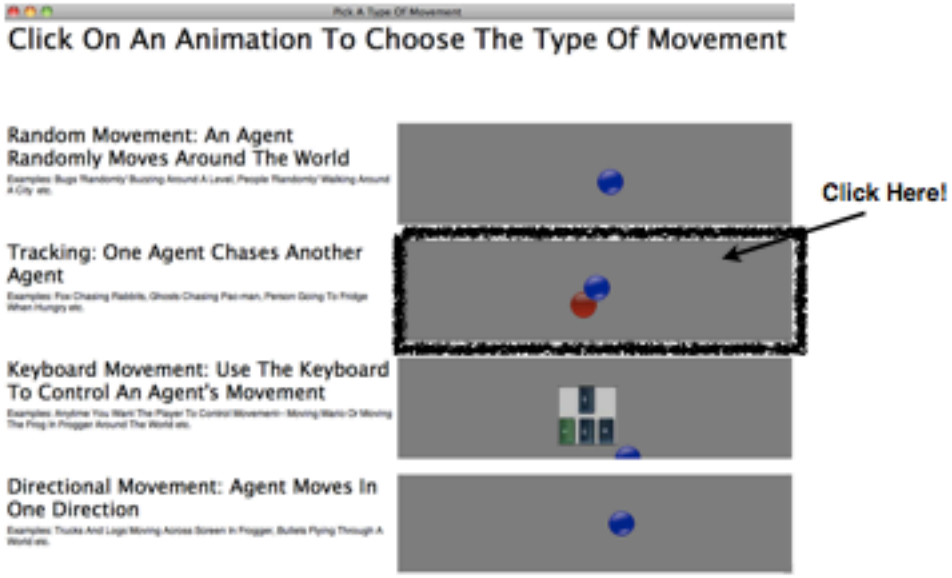
14. Click The Add Pattern Button In The Bottom Left (If You Closed The Following Window You Can Open It By Hitting The Blue Circle Button)



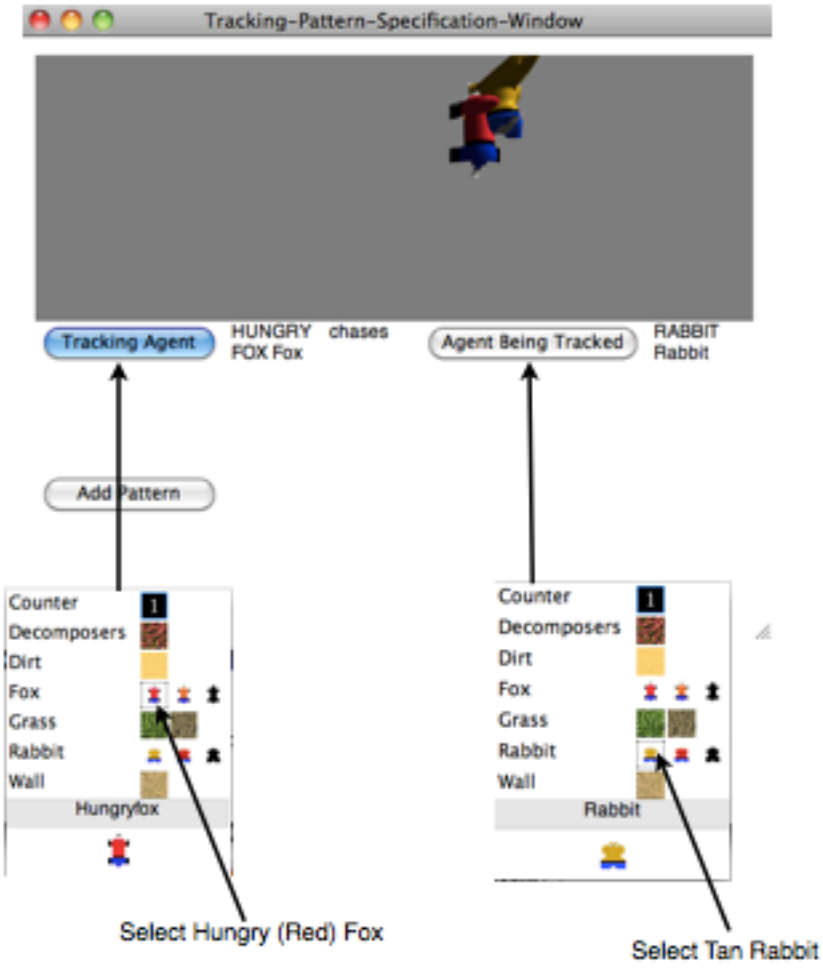
15. Click On The MOVEMENT Animation This Time



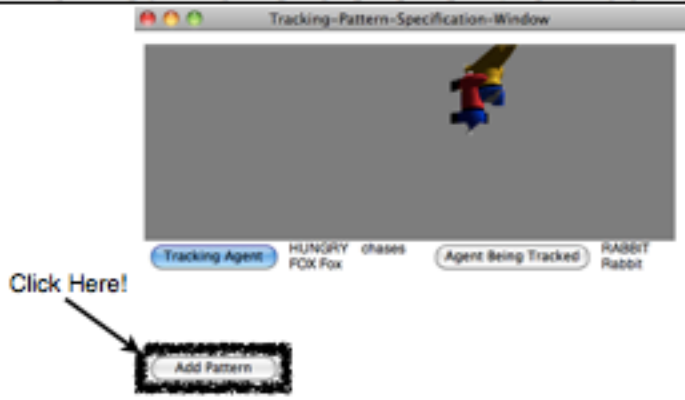
16. Now Click On TRACKING Animation



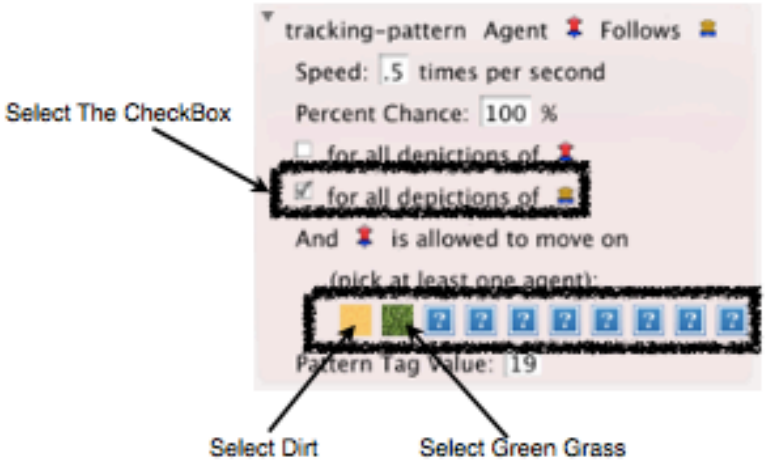
17. We Make The Hungry (Red) Fox Track The Rabbit As Follows



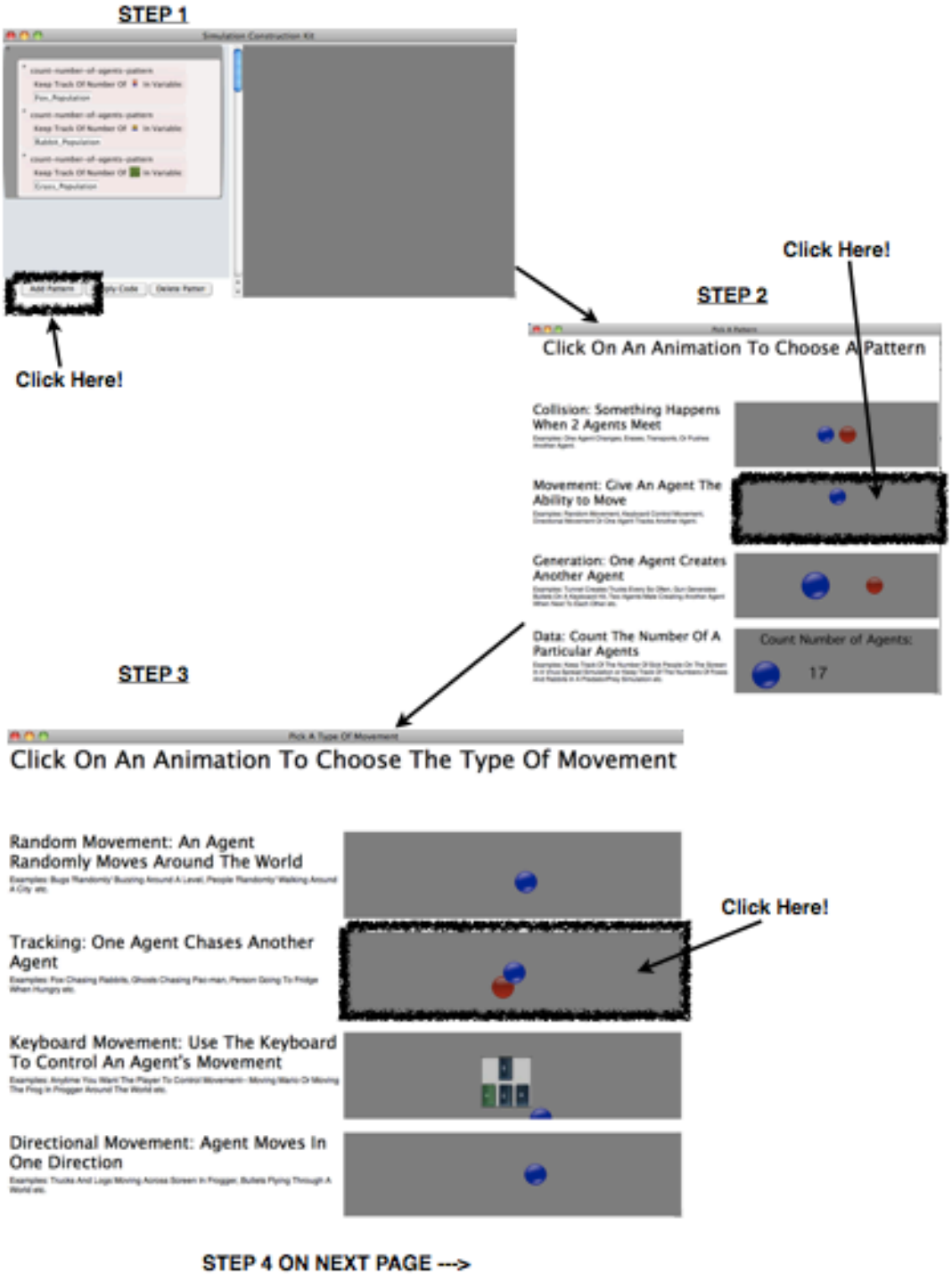
18. When You Are Done Click on the "Add Pattern" Button



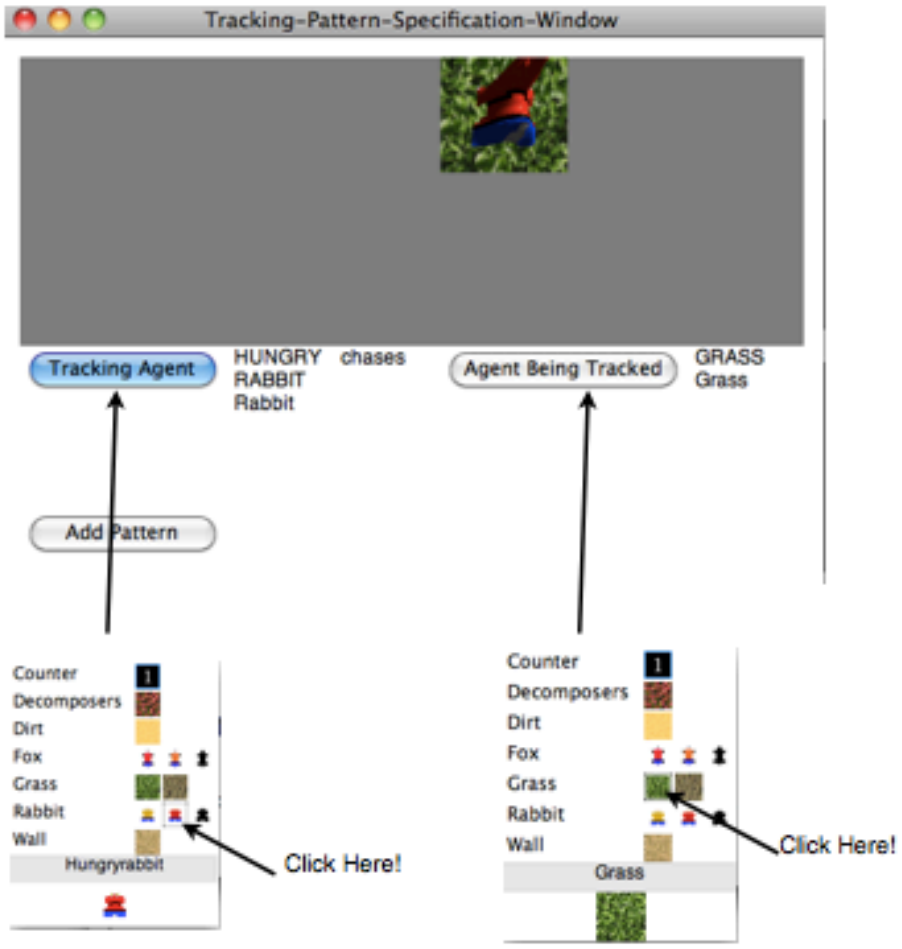
18.We Want The Fox To Chase Any Depiction Of The Rabbits (Including Hungry Rabbits) And Move On Both Dirt And Grass. We Do This As Follows:



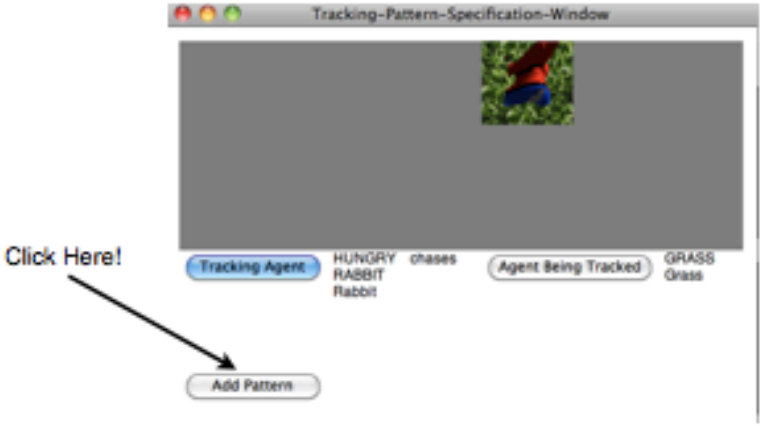
19. Now We Will Make The Hungry Rabbit Go After The Grass. To Do This We Are Again Going To Use The Tracking Pattern Found In Movement. These Steps Are Very Similar To The Hungry Fox Going After The Rabbit. They Are Outlined On the Next Page.


STEP 1
Click Here!
Click Here!
STEP 2
Click On An Animation To Choose A Pattern
Collision: Something Happens When 2 Agents Meet
Movement: Give An Agent The Ability to Move
Generation: One Agent Creates Another Agent
Data: Count The Number Of A Particular Agents
Count Number of Agents:
17
STEP 3
Click On An Animation To Choose The Type Of Movement
Random Movement: An Agent Randomly Moves Around The World
Tracking: One Agent Chases Another Agent
Click Here!
Keyboard Movement: Use The Keyboard To Control An Agent's Movement
Directional Movement: Agent Moves In One Direction
STEP 4 ON NEXT PAGE --->

STEP 4

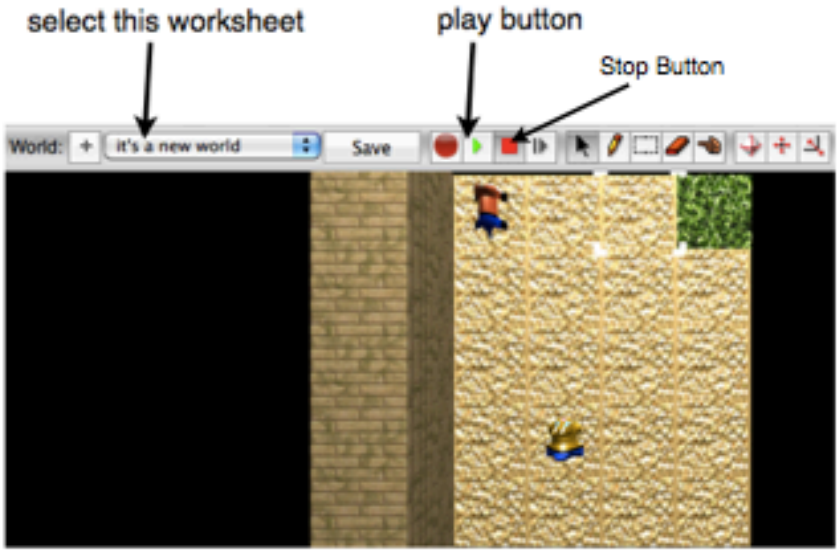


19. Click The "Add Pattern" Button In The Bottom Left



20. At This Point You Should See The Following. Make Changes So That We Allow The Hungry Rabbit To Move On Dirt. DO NOT Let The Hungry Rabbit Move On Grass (The Chasing Won't Work).

Great Job! Now When We Test The Program We Should See The Hungry Fox Chase The Rabbit, And The Hungry Rabbit Run After The Grass. Try It Out! Hit Stop When You're Done.



What Do The Hungry Rabbit And Hungry Fox Still Need To Do In Our Simulation? What Pattern(s) Would You Use To Make This Happen?

_____

_____

_____

C.4 Predator/Prey Worksheet Day 4

WORKSHEET AND VISUAL CHEAT SHEET FOR PREDATOR PREY SIMULATION: DAY 4 (Return After Class!)

NAME: _____

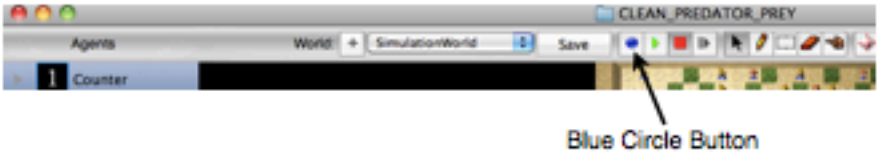
GRADE: _____ TEACHER: _____

PERIOD:_____ COMPUTER NUMBER:_____

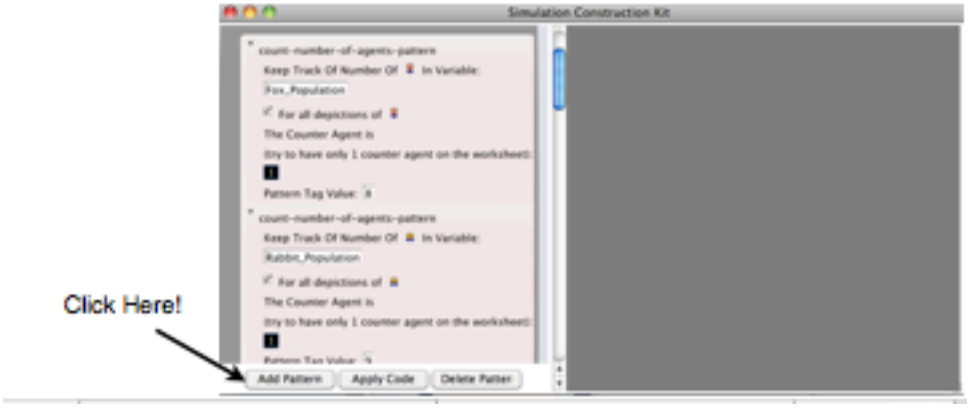
Follow along in class as we show you how to find your simulation you did yesterday and open it.

We Are Now Going To Make The Hungry Fox Change The Rabbit Into A Dead Rabbit

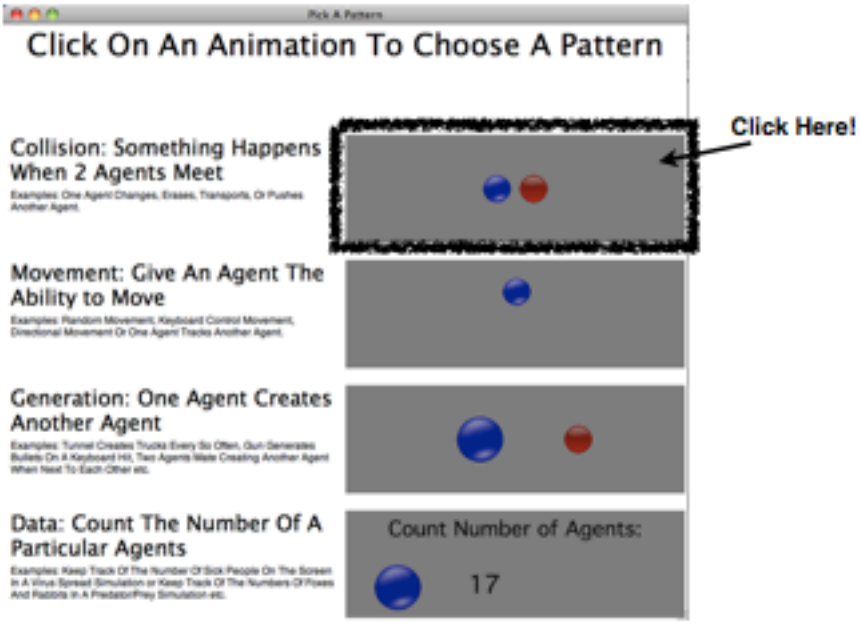
1. Hit The Blue Circle Button. This Button Allows Us To Add A Pattern



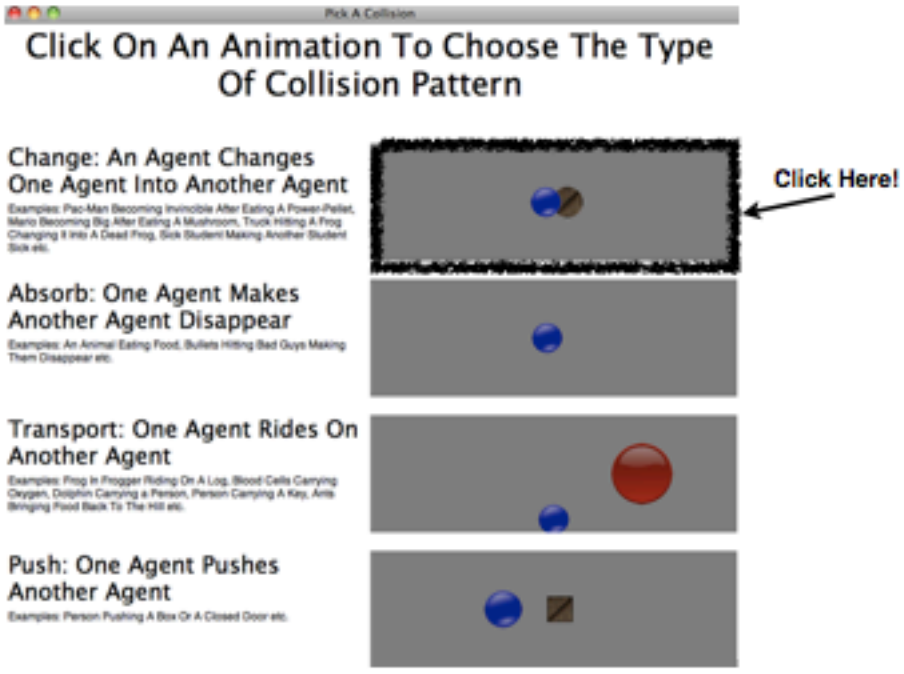
2. Hit The "Add Pattern" Button Located at the Bottom Left



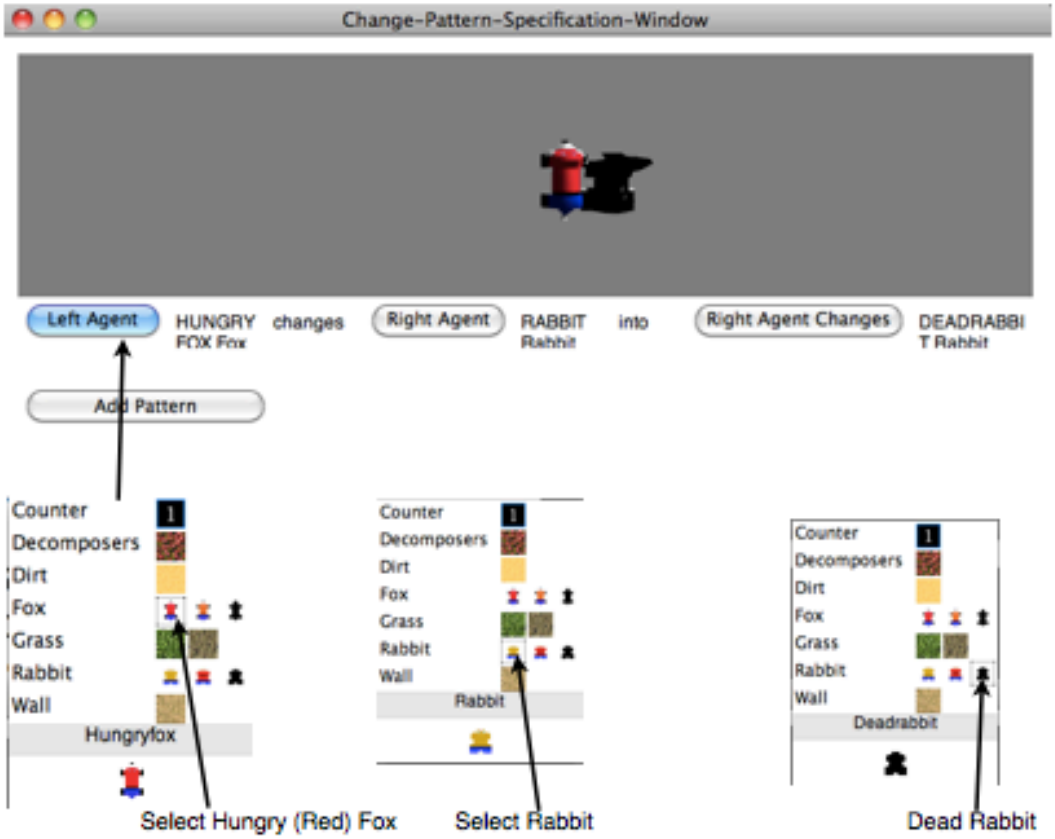
3. Now We Are Going To Click On The COLLISION Animation



4. On The Next Window we are going to click on the CHANGE animation

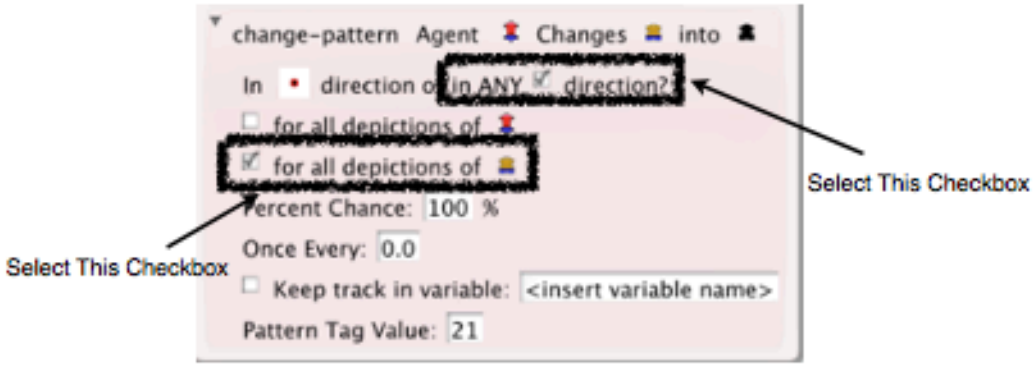


5. We Are Going to Have the Hungry Fox Change The Rabbit Into The Dead Rabbit



6. Click The Add Pattern Button At the Bottom Left

7. We Want the Hungry Fox To Eat All Rabbit Depictions In Any Direction.



We Make The Hungry Fox Eat Any Depiction Of Rabbit So That Regular Rabbits And Hungry Rabbits Get Eaten. What Other Depiction Of Rabbit Might Get Eaten The Way This Simulation Is Programmed? Is This Realistic? How Might You Change This Simulation To Make It More Realistic?

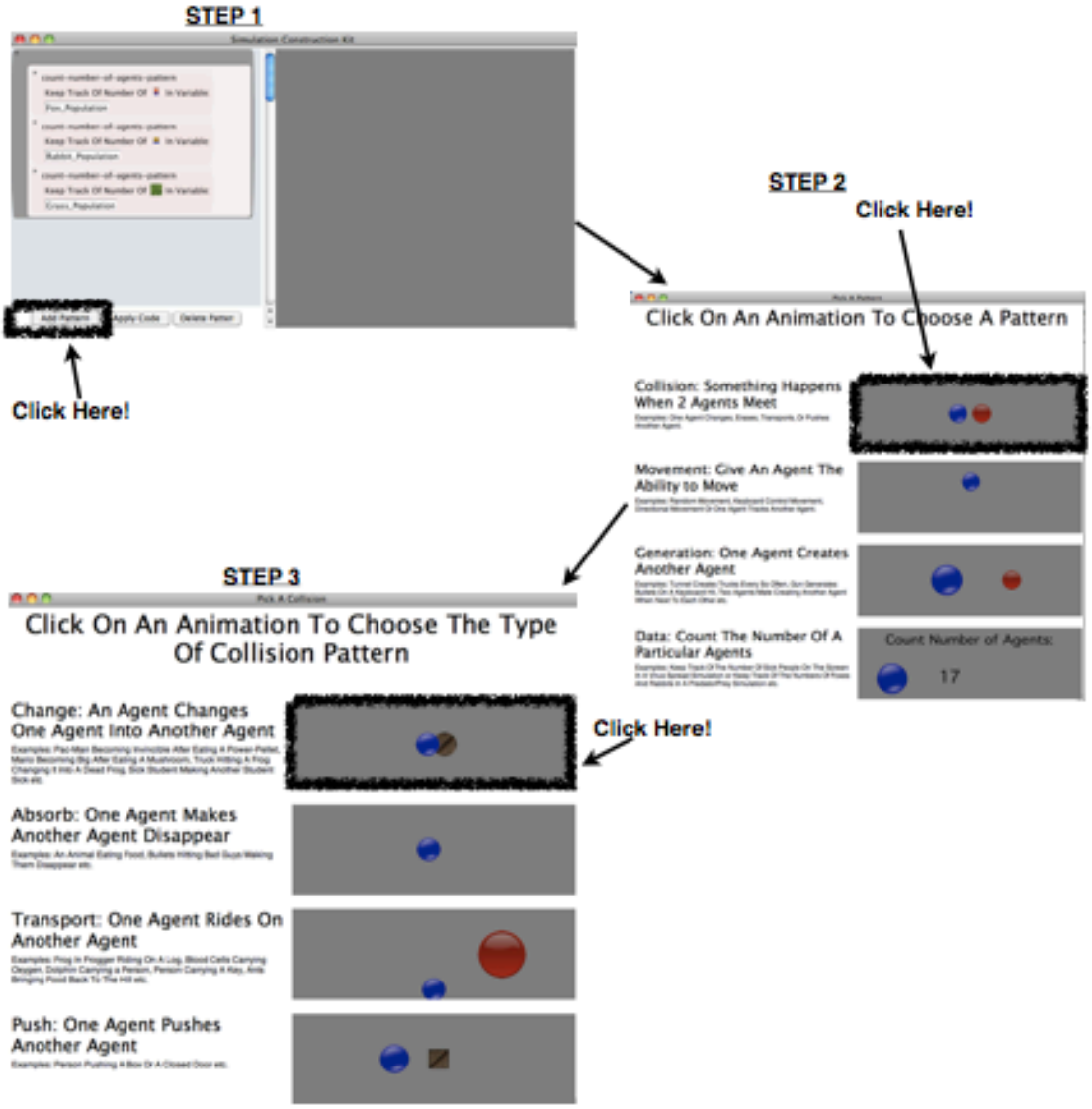
_____

_____

_____

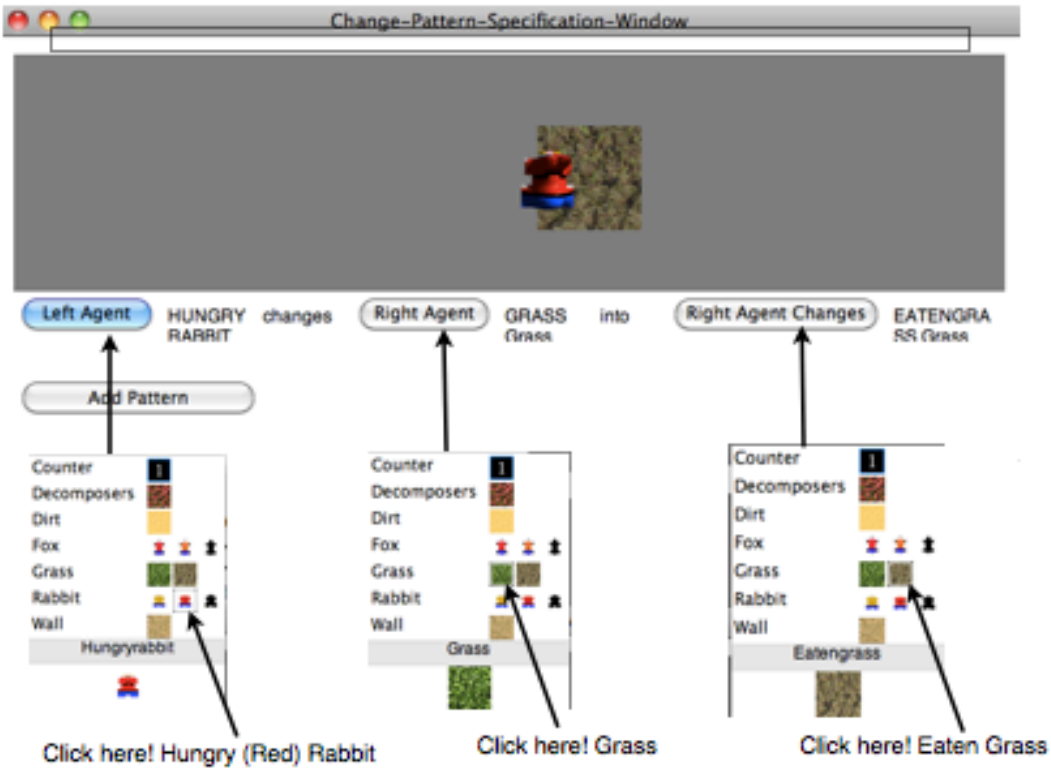
_____

8. We Want the Hungry Rabbit To Eat Grass In Any Direction. This Will Be Very Similar To What We Did With The Fox Eating The Rabbit. The Following Picture Outlines These Steps



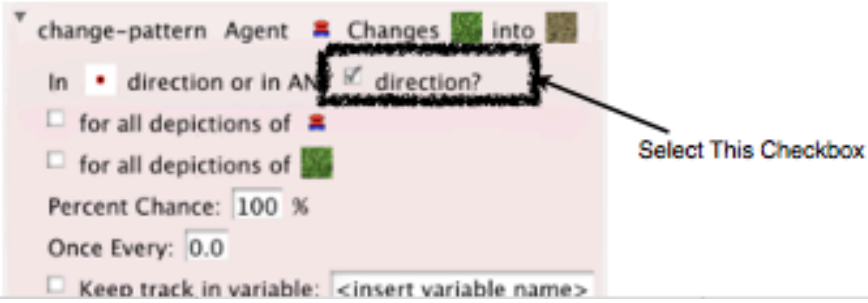
STEP 4 ON NEXT PAGE --->

STEP 4



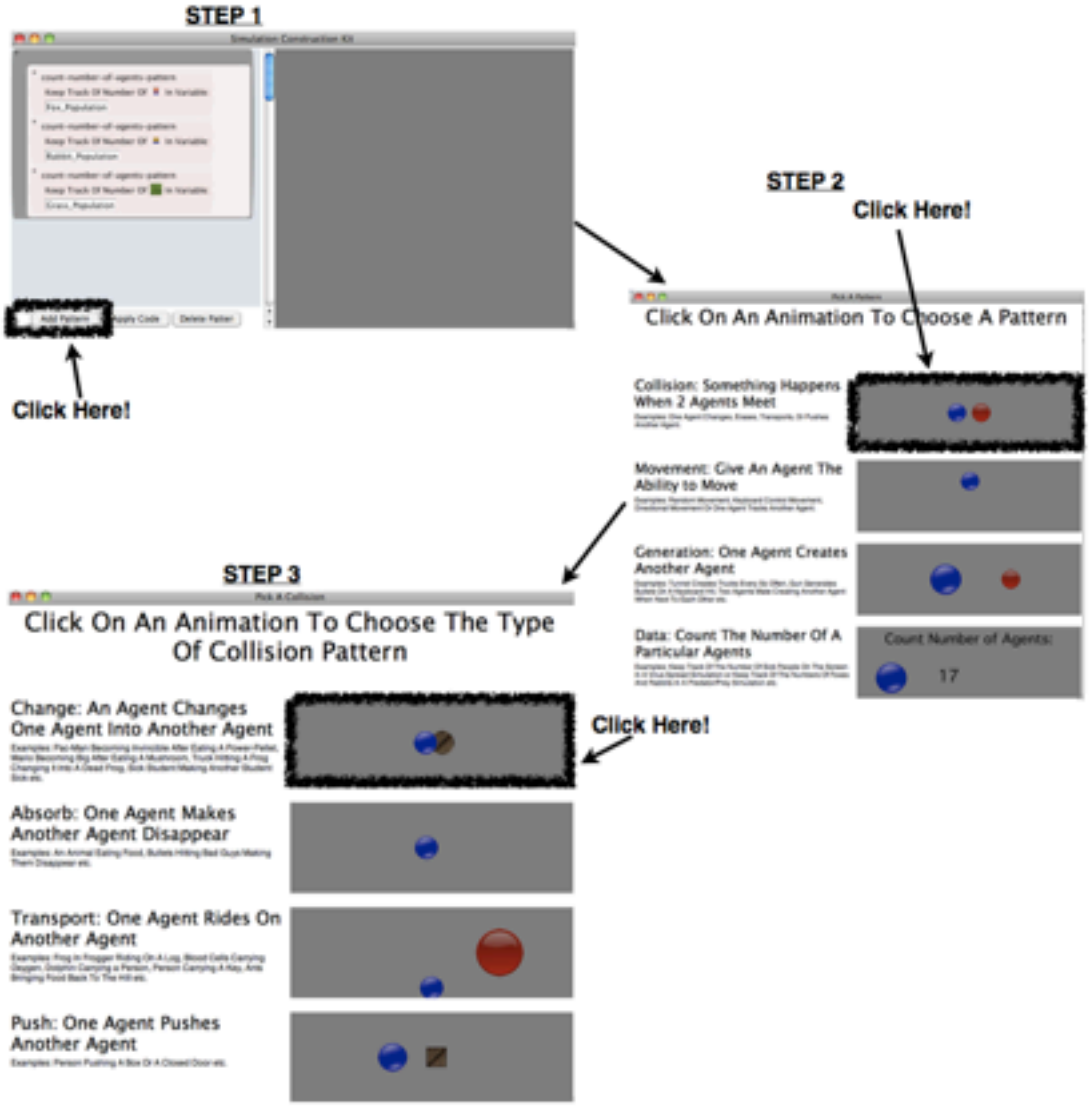
9. Click The "Add Pattern" Button At the Bottom Left

10. Click "Any Direction" In The Selection Box

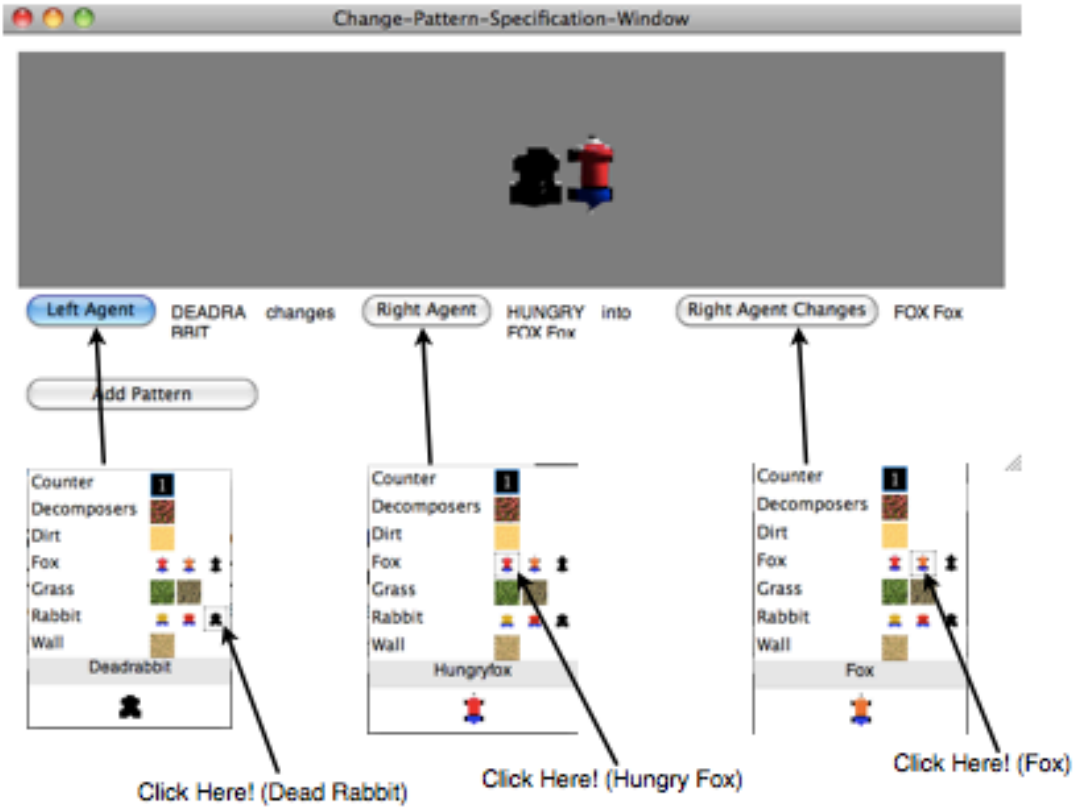


Now We Are Going To Make The Dead Rabbit Change The Hungry Fox Back Into A Regular Fox And The Eaten Grass Change The Hungry Rabbit Into A Regular Rabbit.

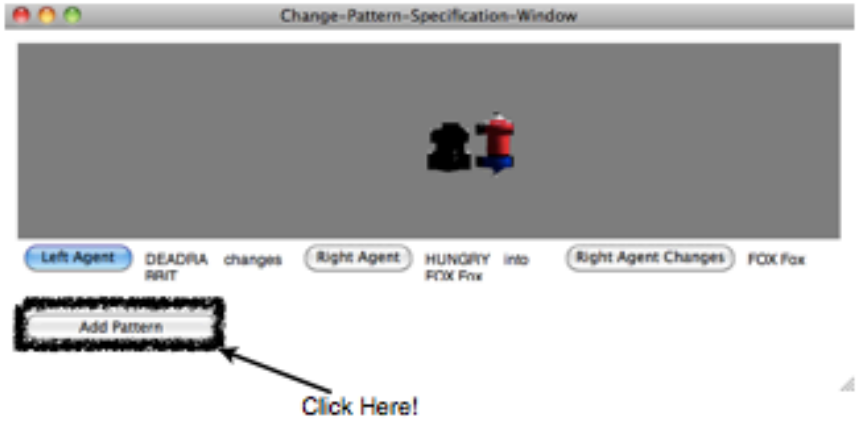
11. First Let's Do The Fox. This Is Similar To What We Just did- Follow The Steps Outlined below.

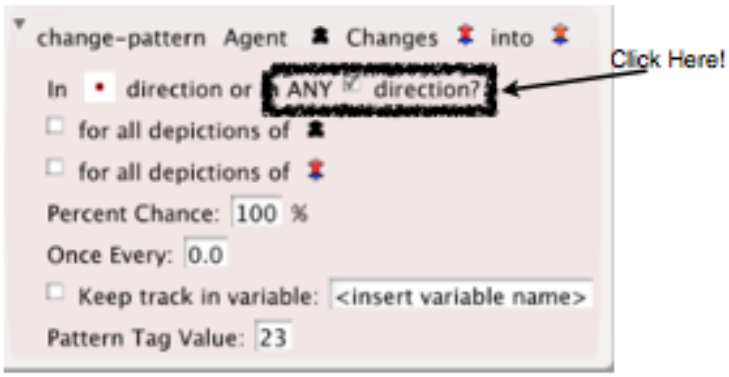


STEP 4

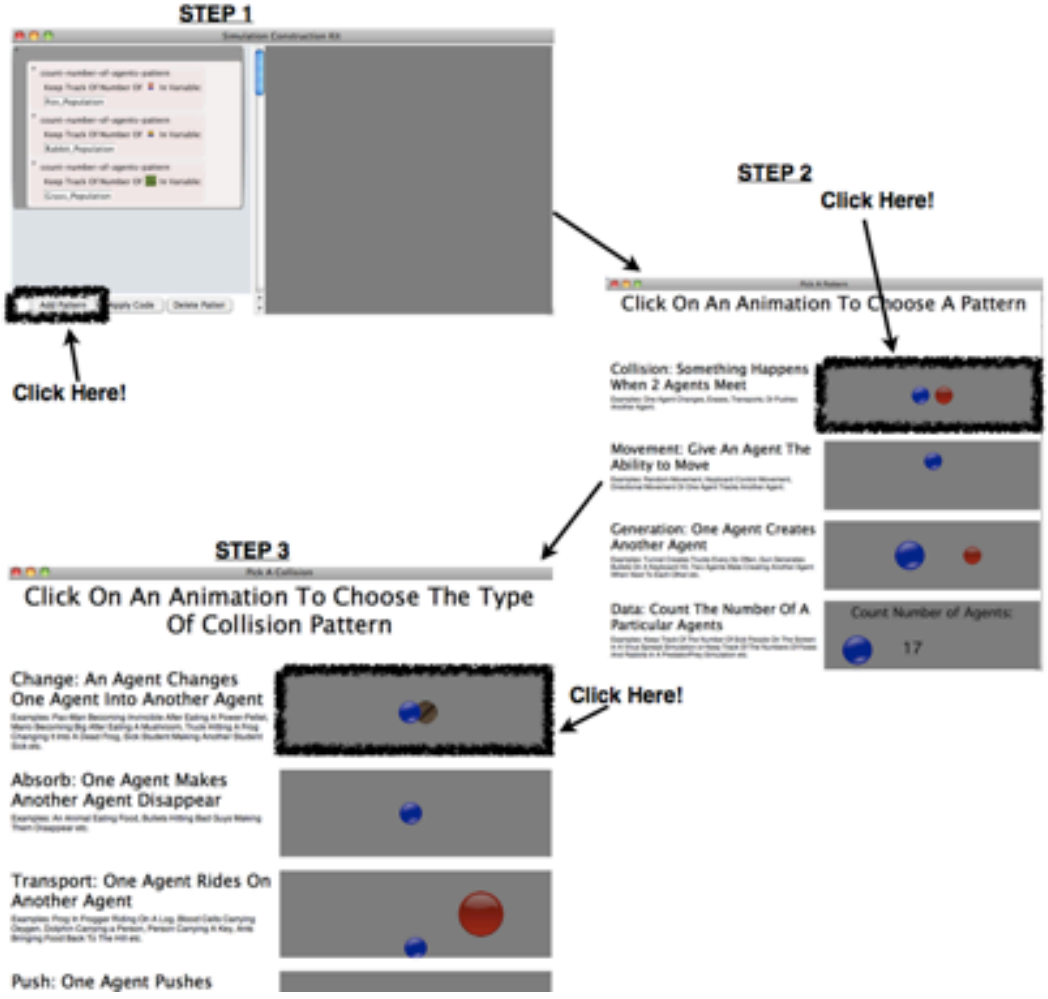


12. Click The Add Pattern Button At The Bottom Left

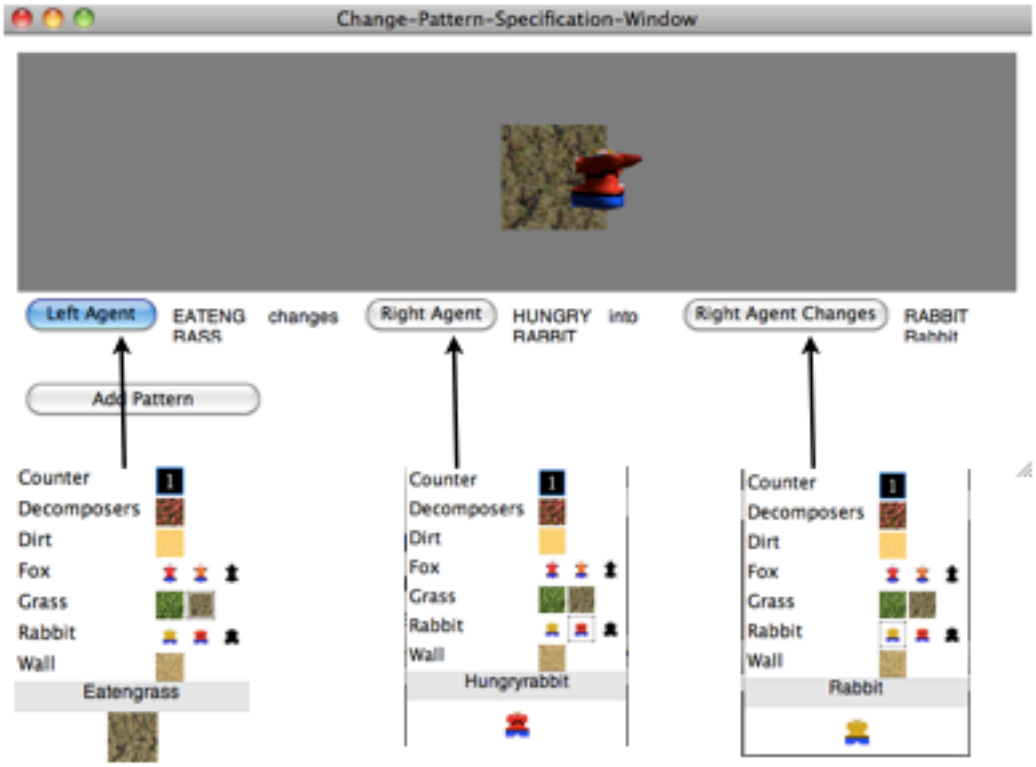




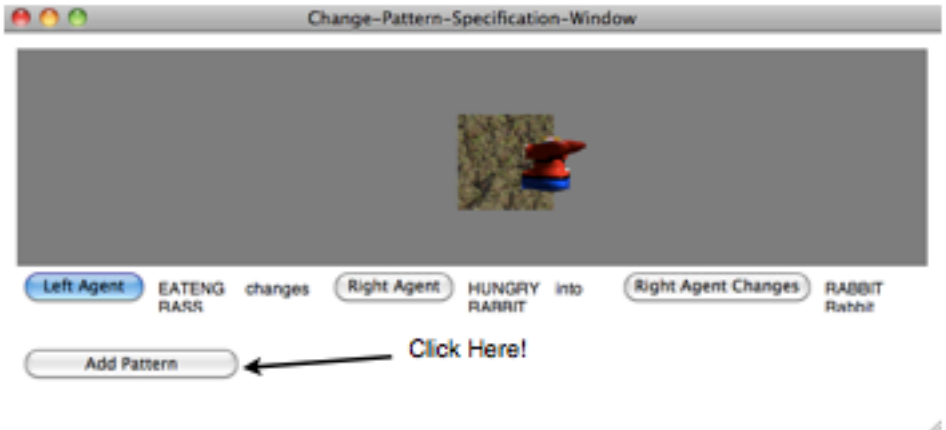
13. Now Let's Make The Eaten Grass Change The Hungry Rabbit Back Into A Regular Rabbit.



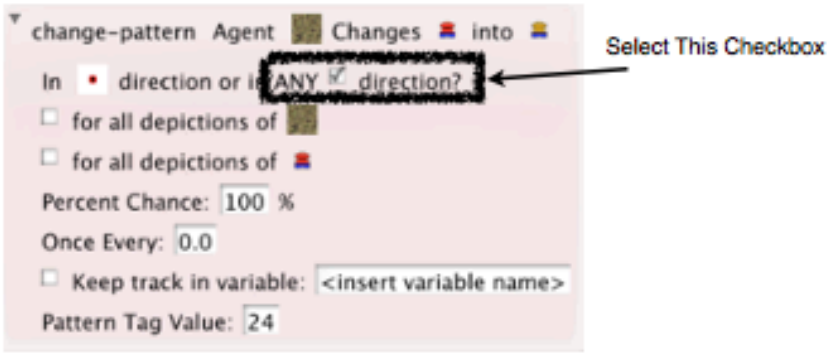
STEP 4



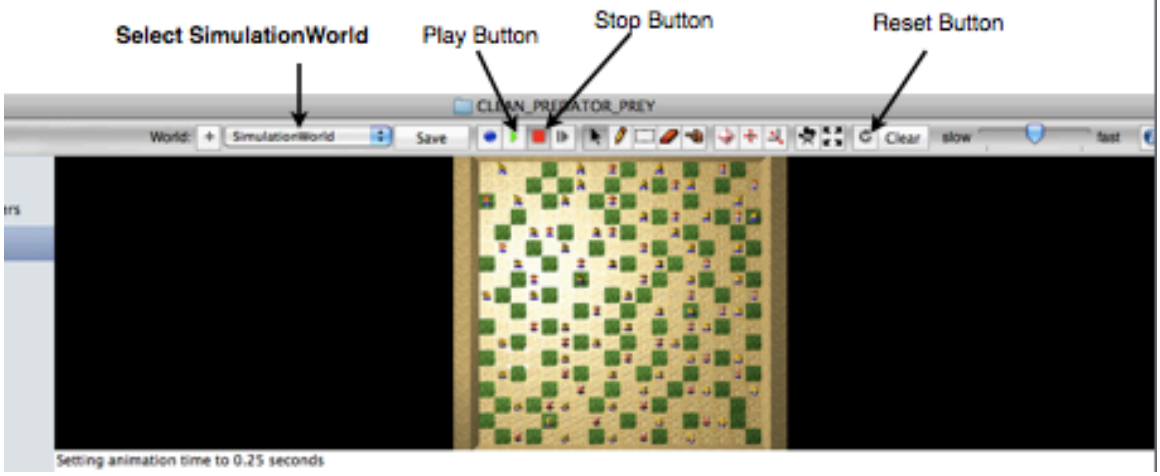
14. Click The "Add Pattern" Button At the Bottom Left



15. Now We Want The Grass To Change The Hungry Rabbits Into Regular Rabbits In Any Direction. We Do This As Follows



16. You Can Test Your Simulation but This Time Do It In The Simulation World. Open Up The Simulation Properties Window And Record The Time It Takes For The Foxes And Rabbits To Die Out On the next Page.



About How Many Seconds Did It Take For Every Fox And Rabbit To Die Out?

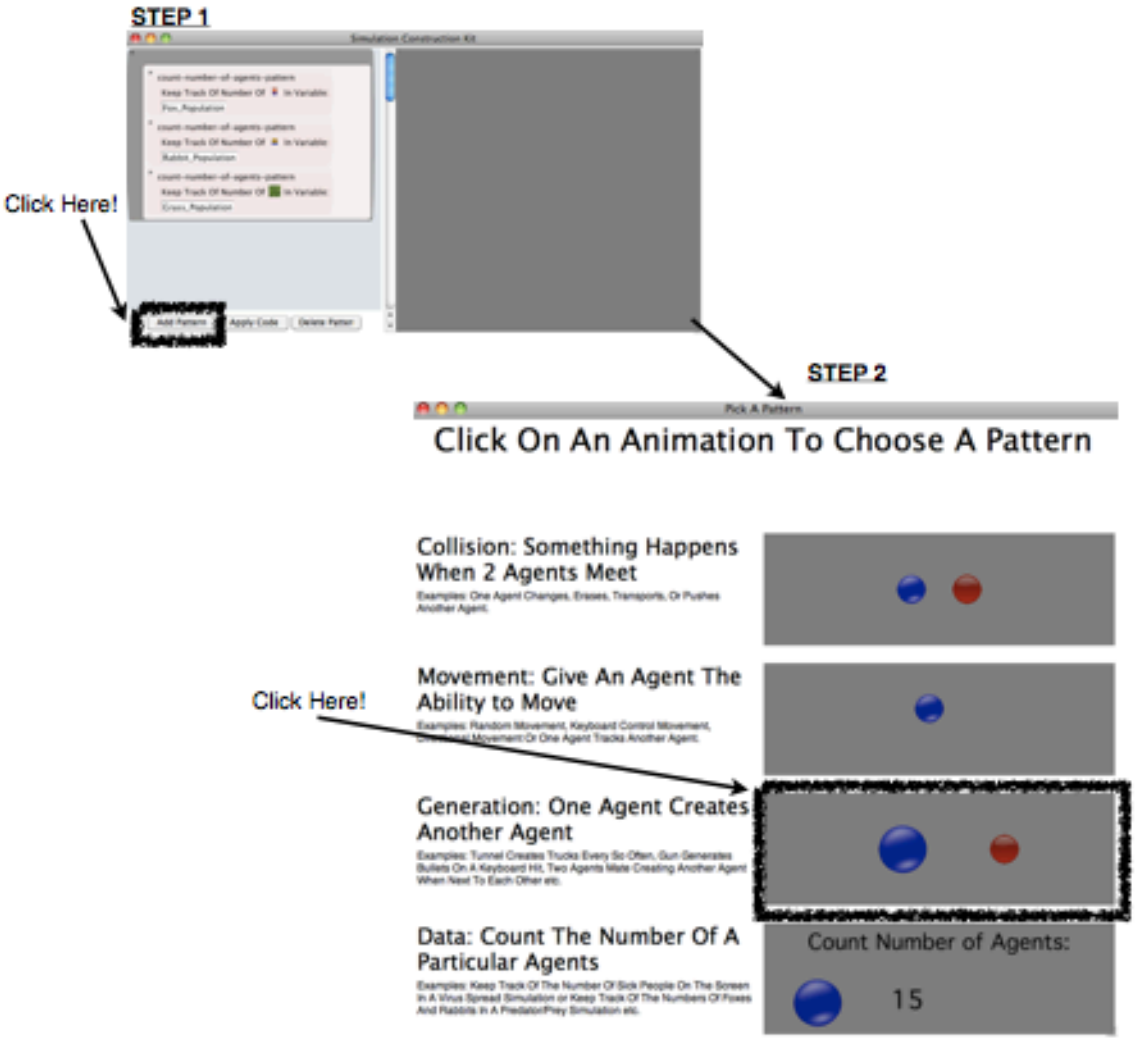
_____

What Pattern Would You Use To Add Mating To This Simulation?

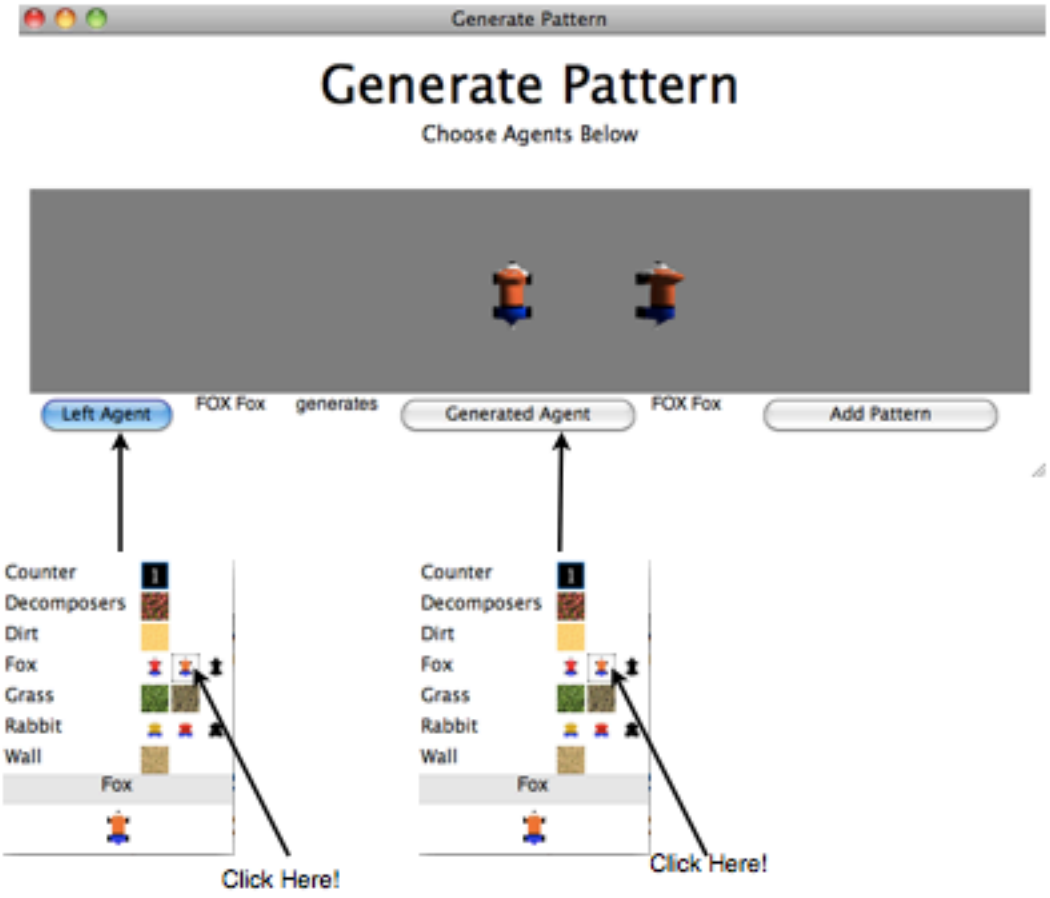
_____

_____

17. Now We Are Going To Add Mating. Foxes Will Make New Foxes And Rabbits Will Make New Rabbits! The Following Steps Outline This.

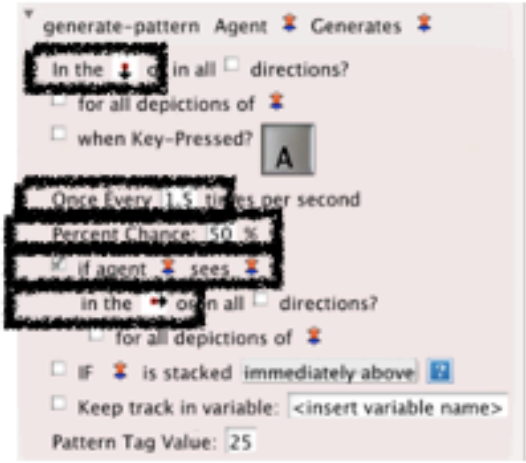

STEP 1
Click Here!
STEP 2
Click On An Animation To Choose A Pattern
Collision: Something Happens When 2 Agents Meet
Movement: Give An Agent The Ability to Move
Click Here!
Generation: One Agent Creates Another Agent
Data: Count The Number Of A Particular Agents
Count Number of Agents:
15

STEP 3

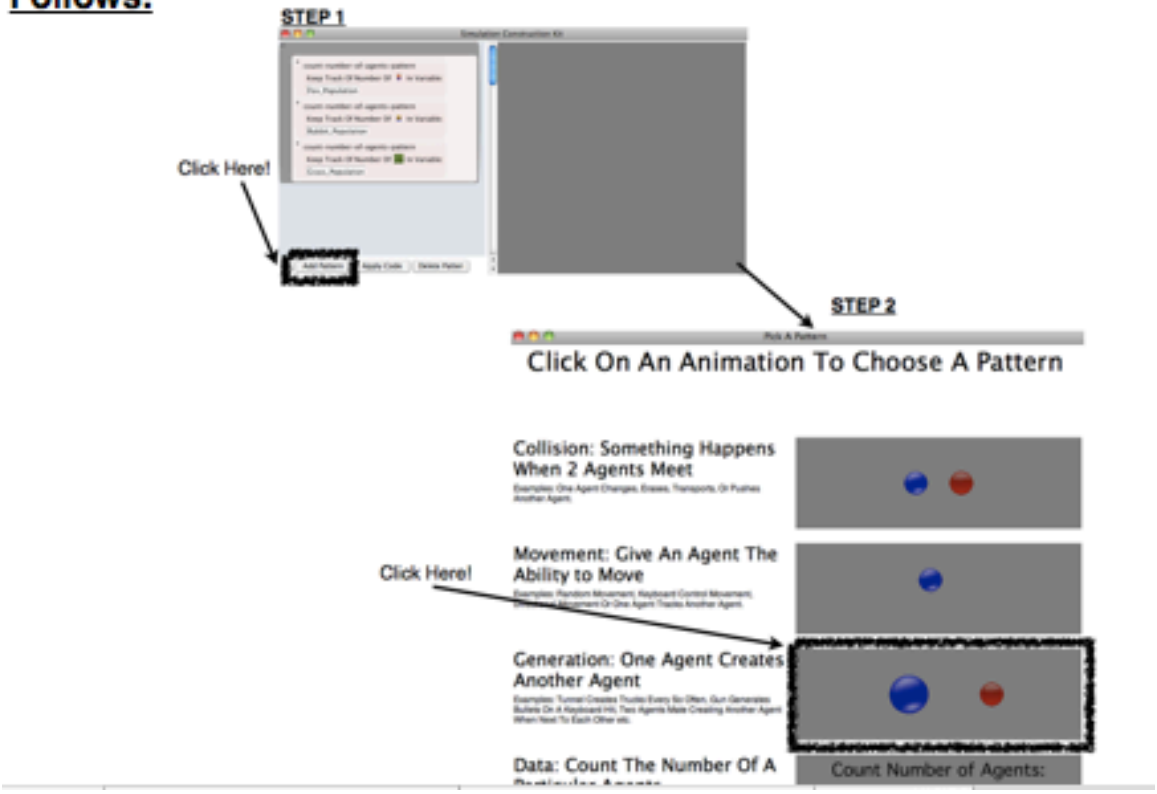


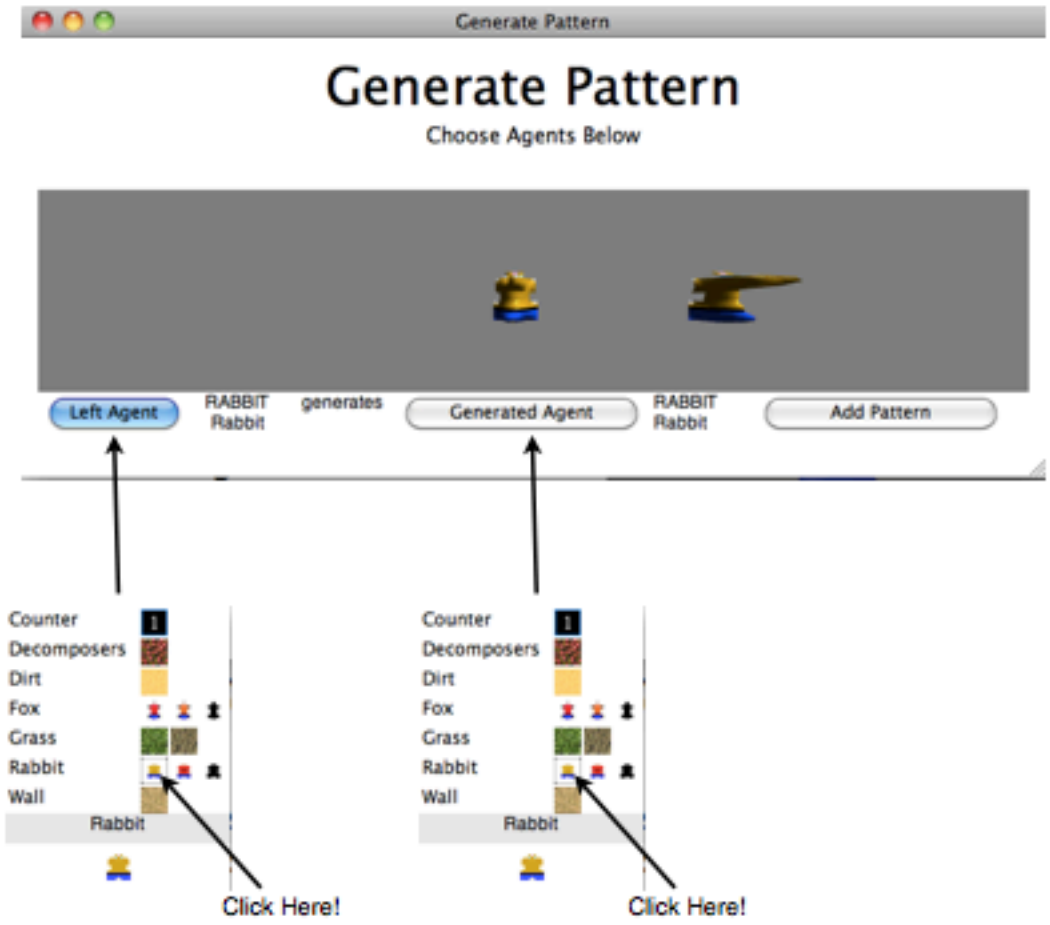
18. Now Click The "Add Pattern" Button In The Bottom Right

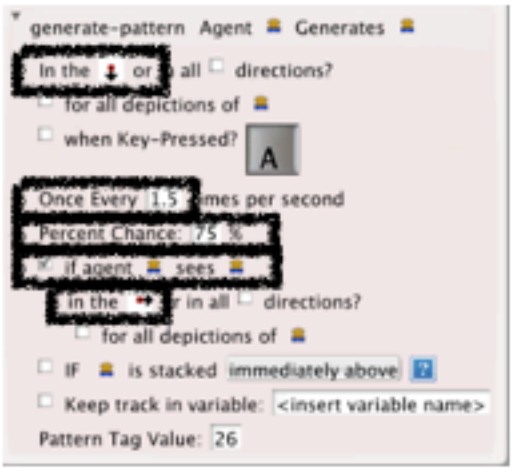
19. We Are Going To Have The Fox Create Another Fox In The Downward Direction If It Sees A Fox To Its Right Once Every 1.5 Times a Second. We Do This By Making The Following Changes.



20. Now We Will Do The Same For The Rabbits. The Steps Are As Follows.




Generate Pattern
Generate Pattern
Choose Agents Below
Left Agent
RABBIT
Rabbit
generates
Generated Agent
RABBIT
Rabbit
Add Pattern
Counter
Decomposers
Dirt
Fox
Grass
Rabbit
Wall
Rabbit
Click Here!
Counter
Decomposers
Dirt
Fox
Grass
Rabbit
Wall
Rabbit
Click Here!


generate-pattern Agent Generates
In the or all directions?
for all depictions of
when Key-Pressed?
A
Once Every 1.5 mes per second
Percent Chance: 75 %
if agent sees
in the r in all directions?
for all depictions of
IF is stacked immediately above
Keep track in variable: <insert variable name>
Pattern Tag Value: 26

C.5 Predator/Prey Worksheet Day 5

WORKSHEET AND VISUAL CHEAT SHEET FOR
PREDATOR PREY SIMULATION: **EXPERIMENTS SECTION (Return After Class!)**

NAME: _____

GRADE: _____ TEACHER: _____

PERIOD:_____ COMPUTER NUMBER:_____

Since You Have Finished Your Simulations We Will Now Experiment On Your Creations To See What We Can Understand! The First Thing We Will Do Is Run The Simulation And Record The Amount Of Time It Runs For The Rabbit And Foxes To Die Off. If You Significantly Updated Your "SimulationWorld" Worksheet (The One With A Bunch Of Foxes And Rabbits On it, Call Me Over And I Can Put The Old Worksheet Back On It).

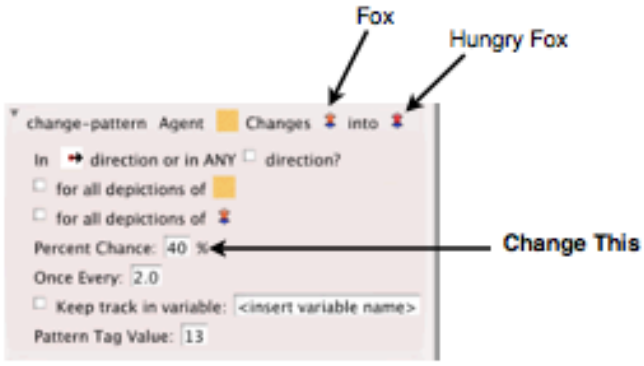
Feel Free To Work With A Classmate On This But Please, Each Of You, Fill Out A Worksheet!

Open The Simulation Properties (Window Menu At The Top-->Simulation Properties) and Hit Run On The Simulation. How Long Does It Take For Either The Rabbit Or Foxes To Die Off? (Give The Time Of The One That Dies Off First)

Time It Takes To Die Off:_____ seconds

Adjust The **Percent Chances** For One Of The Following: Mating Rate (Generation), Hunger Rate (Dirt Changing Fox Or Rabbit To Hungry Fox Or Rabbit), Or Death Rate (Dirt Changing Fox Or Rabbit Into Dead Fox Or Rabbit) To Try To Increase The Time It Takes For Either The Fox Or Rabbits To Die Off.

Fox Example If You Picked "Hunger Rate" Change This Percent For The Fox And/Or Rabbit (If You're Confused Call Me Over):



What Did You Decide To Change?

_____

_____

Why Do You Think This Will Increase The Time For Either The Foxes And/ Or Rabbits To Die Off?

_____

_____

_____

Run Your Simulation With The New Percentages That You Put In. What Happened (Did The Simulation Time Increase Or Decrease?)

_____

_____

_____

Why Do You Think You Got This Result?

_____

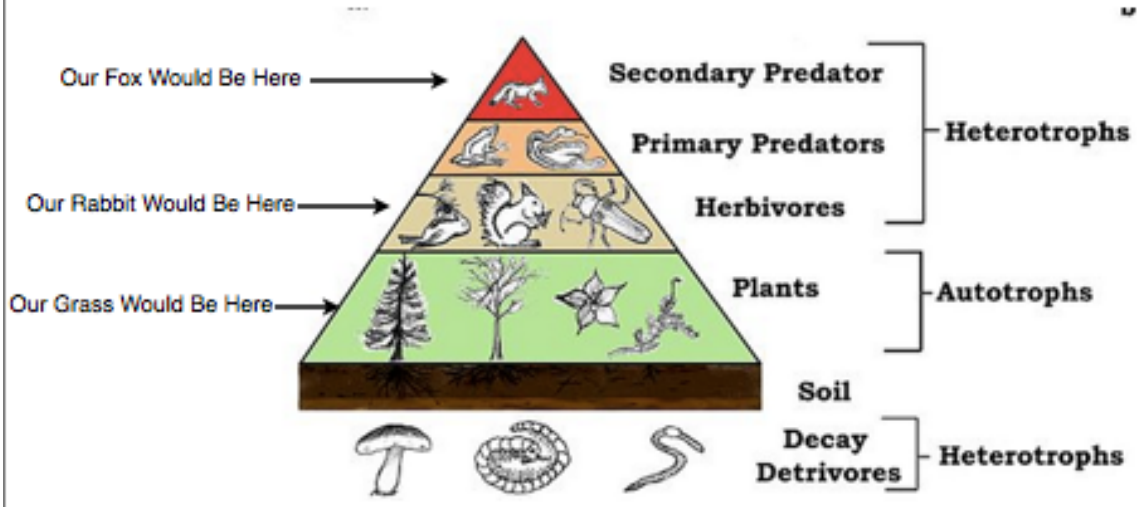
_____

_____

Try A Few More Values. What Got You The Best Result (Longest Time)? What Was That Time?

Let's Say Our System Reaches "Equilibrium" if the Foxes And Rabbits are able to maintain a certain number living (ie: The Dead Foxes And Rabbits Are Replaced By New Ones) So The Foxes And Rabbits Never Die Off. In This System, Is It Hard Or Easy To Keep It In Equilibrium?

A **Trophic Pyramid** Is Visual Representation Of The Biomass At Each Level Of An Ecosystem.....
The Following Is What A Trophic Pyramid Might Look Like (from https://en.wikipedia.org/wiki/File:TrophicWeb.jpg)



Let's Calculate The Biomass At The Top Two Levels Of Our Pyramid At The Beginning Of Our Simulation

Let's Say Each Fox Is 15 pounds. We Have 36 Foxes To Start. How Many Pounds Of Biomass Do We Have At The Fox Level (Type it into Google if you need a calculator)?

Secondary Predator Biomass _____ pounds.

Let's Say Each Rabbit Is about 4 Pounds. We have 54 Rabbits To Start. How Many Pounds Of Biomass Do We Have At The Rabbit Level.

Herbivore Biomass _____ pounds

Given The Pyramid Picture Above, Is Our Simulation Realistic? If Not, How Might You Change It?

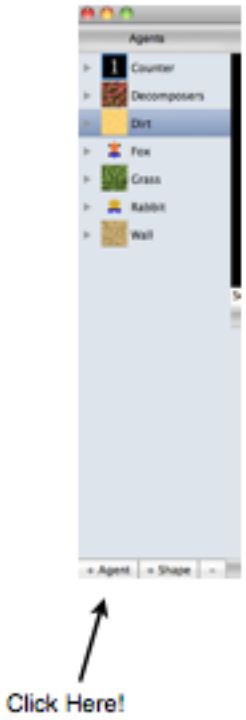
_____

_____

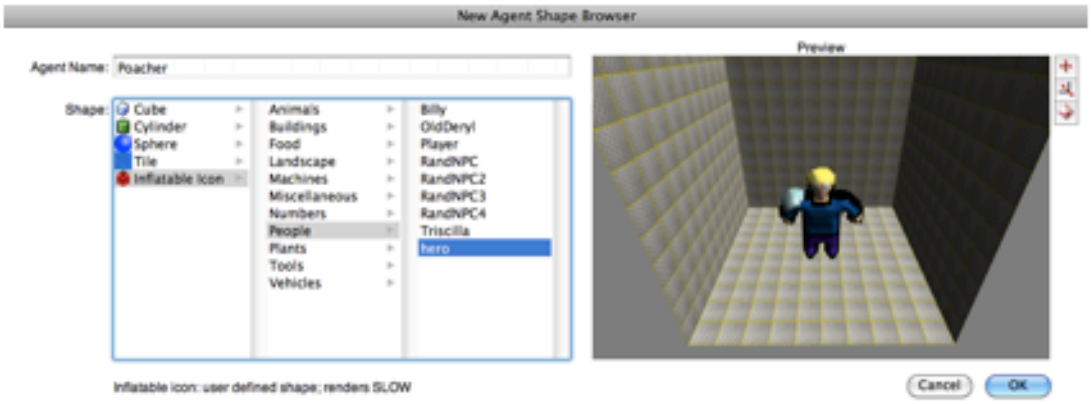
_____

Let Us Add A Poacher To The Simulation. To Do This We Will Add Two Agents: A Poacher And Bullets. The Poacher Will **Generate** Bullets, The Bullets Will **Absorb** foxes, And The Bullets Will Use **Directional Movement**.

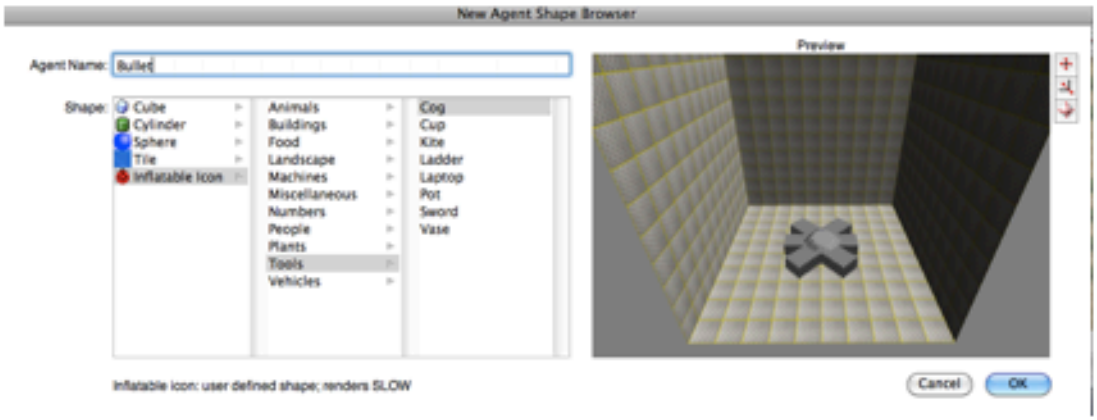
First ADD An Agent



The Following Picture Shows You How To Add A Person To Your Simulation. Click "OK" At The Bottom Right When You Are Done. Call Me Over If you Need Help.



Now Add A Bullet, I did mine as follows (I used the "COG" icon). Hit OK Again when You're Done.



Now Make The Poacher **GENERATE** the bullet in a given direction.

Make the Bullet use **Directional Movement** to move.

Make the Bullet **Absorb** The Foxes In that Same Direction.

Call Me Over If You Need Help.

Finally Put Some Poachers On Your Worksheet and see if they make the Foxes Disappear.

Does The Introduction Of Poaching Make It Harder Or Easier To Keep The System In Equilibrium? Explain Why Or Why Not?

_____

_____

_____

GREAT JOB! Thank You For All Your Help. If You Are Done Feel Free To Play Around In Agent Cubes. You Can Even Open A New Project And Create What You Want.

APPENDIX D: Teacher Diaries

D.1 Teacher Diaries April 16-April 19. 7th Grade Life Science.

Teacher Diary Monday April 16

Today I started by introducing Patterns, Doing a C.S. unplugged exercise to try to link the patterns to the predator prey model ~~I wanted to~~ we are creating, and then showing them ~~what~~ the system is like and introducing them to Agent cubes and the pattern picker before letting them start the tutorial with the remaining time. I was pretty anxious because there is a lot to do and not a lot of time (4 days ~ 50 min each day ≈ ~~45~~ 40-45 min after introductions). Today was even shorter because Marco had to spend more time bringing the students who missed last Friday up to speed (there was a lot of students absent because of a field trip). This led to not much time for students to actually get started. Period 2 I tried to walk them through each pattern but each student seemed to be confused because of how fast we were moving. Furthermore, It was a full class & hard to get everyone's attention (partly because it's hard to remember how to teach on the 1st class with 6th graders). Also I didn't link the patterns to the simulation well enough. For period 3 ~~use~~ I gave them more time to work on the sim on their own after making the essential introduction points (I also had to show them where the sim was & name them etc.). I was way more active helping the students because I wasn't walking them through every step -> so I could help the students who were having trouble more 1 on 1. This was a way better teaching strategy and the students responded w/ many of them jumping past day 4 into day 2 (The lessons are split up into 4 days but really I want them to finish in 3 days so they can run experiments on day 4). Overall a good day but am wondering if everyone can finish.

Teacher Diary Tuesday April 17

Today went awesome. The kids got in early, Marco basically gave me the whole period. Also everyone seemed excited to do simulations which was a plus, and everyone was super attentive. I didn't try to throw too much at them for my introductin; I just talked about common problems students were having, reviewed the day before, talked about what were doing today & let them go. So students had most of the period to create their Sims. Period 2, the one I was concerned about, did great w/ many kids getting to day 3 and a few kids finishing! Many kids also finished in the other periods (3, 5, 7) which is fantastic. Tomorrow I am going to have them experiment on their simulation --> vary mating parameters, vary death rates, hunger rates, add a poacher, calculate energy @ each trophic level etc. This is really what I wanted to show w/ the system that students could easily create sims & then gain understanding by actually doing. I also want to ~~push~~ make these activities now scaffolded so students have to figure things out more. ~~One~~ The only downside of today were bugs => we get a "spinny wheel of death" if students try to add a pattern while a sim is running (and other times it seems) which is annoying. I feel bad & I have to restart the program, but usually everything is saved and it all works after which is good. Also the kids seem to be having so much fun that they all don't mind it so much. But still, in one class we must have had 6 of them! I think in a program like mine which integrates with another Beta program doing a bunch of stuff that wasn't originally intented, it can easily break things. Most of them had to do w/ a sim running & adding patterns which makes me think it has to do w/ the system expecting something to be selected in behaviours but my program selects something else which leads to crashes. Other than that one setback, fantastic day!

Teacher Diary Wed April 18

Today went great - I started by giving a brief introduction talking about the "days" experimental worksheet. Then I gave them time to do their creatims. Today was definitely tiring - I'm not used to teaching 3 days in a row. The students, on the other hand, seemed very engaged & were creating their simulations & understanding as they went along. A few people even finished the poacher (last part of the experiments section). One kid said that it was not unrealistic for a fox to become hungry fix to become unhungry around a dead rabbit because it could still eat a dead rabbit. Though I don't think a fox would really eat an already dead rabbit found in the wild it showed he was thinking critically. This seemed to be par for the course w/ most students. I wouldn't be surprised if most students finished the simulation tomorrow & had some if not all the experiments done. Period 2 still seems behind but some kids were even finishing in that. & students were given minimal direction on how to implement the poacher & Bullets & Mary completed it! There were some hiccups w/ generation pattern not keeping its data but it was a clean fix, just gotta remember to show the kids tomorrow. There were some spinny wheel of death issues but way less, maybe 1-2 per class (mostly due to excessive generation). Kids seemed to be having a blast doing the experiments & I heard overheard comments like "this is way better than putting in code" which is kinda cool. & Other students having problems was because they made quick mistakes => But it must be said that given the chance to do this again I would have a way to debug it or at least make it obvious when an unimplemented pattern ends & another pattern begins to help w/ troubleshooting. Of course this is something I would have only discovered through this study. Hopefully tomorrow is laid back w/ people who are done exploring & others finishing up?

Teacher Diary Thursday April 19

Today ended up great. I was a little concerned because it was the day before a long weekend & Marco wasn't there. But the kids all seemed to go through their work & tried their best to finish what they could. With 15 minutes left in the class I went around and dragged students games into their flashdrives (to ensure they were there). Thus, like the first day the students had much less time to work. Some students were not their because of the long weekend - they might have been taken by their parents on vacation etc. Still, the ~~people~~ students who were there ~~were~~ all seemed to work diligently. The substitute teacher let me run the class and after one of the classes came up and talked about how cool this was for the kids. She was in GIS and wanted to create a GIS unit for Boulder high. At the end, the students who had finished all seemed to use the pattern maker to implement various aspects of their program. Some kids turned their sims into a game - I tried to make them not mess w/ their sims in general. Overall students seem to have fun w/ the whole experience w/ many understanding the experiments section ~~of~~ & w/ little to no guidance. It's unfortunate I can't use this system over a whole semester because now students can create sims as the semester goes on much easier the next time around ⇒ I estimate 1 day for sim creation & 1 day for experimentation (Most sims don't have to be this complicated - we can explore 1 or 2 concepts deeply w/ a much simpler sim) The other promising thing that happened was students were able to debug their projects ie: I would say your mating doesn't look right & they would laugh because they had a fox mating w/ a rabbit etc. This is promising for possible future system incarnations.

Great end to the week!

D.2 Teacher Diaries April 26, 27, and 30. 6th Grade Life Science.

Teacher Diary Savs Day 1 April 26

The first day was eventful. Savs told me that his 6th period class would be a handful (he has 5th & 6th period) because they were all the "tough kids" (because of tracking they put all the tough kids together). The day before we went through the beginning lesson => making the Fox move randomly, so he could help out in the class. The difference b/w the 7th and 6th graders became apparent right from the get-go. whereas I had done simulations w/ the 7th graders, when I asked the 6th graders about simulations only a few kids raised their hands. Furthermore, when I asked about predator/prey models, again only a select few raised their hands (as opposed to the 7th graders that had done an evolutionary predator/prey earlier in the semester). So w/ the 6th graders I think the computer science unplugged exercise was immensely helpful. Similarly, none of these students had Agent Sheets before so I had to explain what an Agent was and what behaviors were => But I just talked about behaviors as high level patterns. After that we got their computers setup w/ the clean predator/prey sim & we went through the 1st lesson before letting the kids start. 5th period seemed to do great off the bat and many were able to finish Day 1. Like savs indicated 6th period was more challenging. Mark took 5 of the troubled kids and put them @ the lower level around him to help them while I roamed the top row. For the most part 6th period seemed to finish Day # 1 too. I do wonder to what extent they are just going through the walkthrough motions as compared to period 7 & But it's hard to say. Getting a simulation of a phenomena you haven't studied w/ a tool you've never used seems like a tall order to begin w/.

Teacher Diary Says Day 2 April 27

Day 2 went Much quicker and we got a lot done. I did about a 7 minute intro on patterns & specifying patterns and let the students get to work. The students all seemed to be really productive - unlike the 7th graders though it didn't seem like the 6th graders thought the practice was "easier!" This was most likely because they hadn't had agar sheets experience before. In this context you really don't get why diffusion is so tough to implement. However whenever students asked questions I was able to get them to debug on their own which was heartening. Not so heartening was the fact that many students didn't get simple things like the correct blocking agar in Agaricubes correct. 6th graders seemed to have less of a grasp on a variety of how things are implemented. I also think the things I assumed they would know based on what the 7th graders know did this group no favors because the knowledge of the system & Predator/Prey sim in general ~~was~~ is completely different. Given the opportunity to this again I would spend more time w/ the introduction & patterns more in depth. Part of the problem is I only have 3 days so it's hard to get much of anything going w/o going kind of quickly @ the beginning. Again Period 5 seemed to be on top of it, very good work and all the questions seemed good. Period 6 was wierder, a lot of students forgetting where they were (re implementing random movement twice etc.). I also had the worksheets disorganized so it took us 2 minutes to get the correct worksheets to the kids which wasn't the best thing for their work. Finally a USB drive got stolen - kinda surprised it took this long => I can get the data from backup, No big deal.

Teacher Diary says Day 3 April 30

Seperating classes by a weekend probably wasn't the best idea :). students had to get caught up on what they did, there was a lot of reimplementation of patterns. I would say that the tool has to have a better gui to differentiate patterns that were created, especially in such a complicated simulation. Pattern reimplementation isn't necessarily bad, just sometimes leads to wierd sim behavior (Foxes moving twice b/c of 2 random movements, bunnies jumping on walls b/c one movement doesn't have blocking agents). Many students actually missed patterns! I was amazed by this and am not sure how this could happen unless they picked up the walkthrough after the weekend and since it all looks pretty similar, decided "hey I've already implemented that". There was a lot of getting out old worksheets as a result ⇒ it might have been a good idea to have them circle the step they were on to keep them @ the right place. Fortunately, since we're @ the pattern level it almost doesn't matter in what order they implement patterns. Many students actually got to the experiments section which was wonderful to see in 3 days! I put those kids behind the 8 ball and they w/o any Agent sheets or context and they were able to get through it w/ a correctly working sim in 3 days! Of course the matting was a point of comedy for the kids. One kid increased his mating %, saw all the rabbit stuck and started cracking up and saying "Those are some busy rabbits!" to the delight of his neighbors. Although Day 4 because of the weekend (to keep kids focused) but really may kids went well beyond what I initially thought.

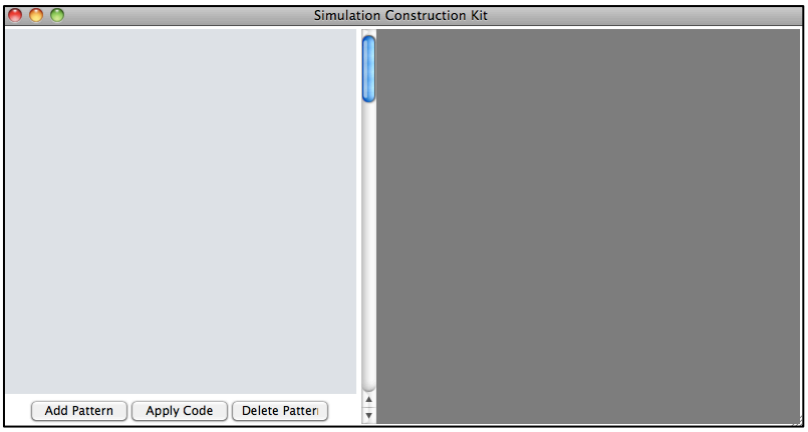
APPENDIX E: Analogical Reasoning Study Materials Provided To Participants

E.1 Introduction To The Simulation Creation Toolkit

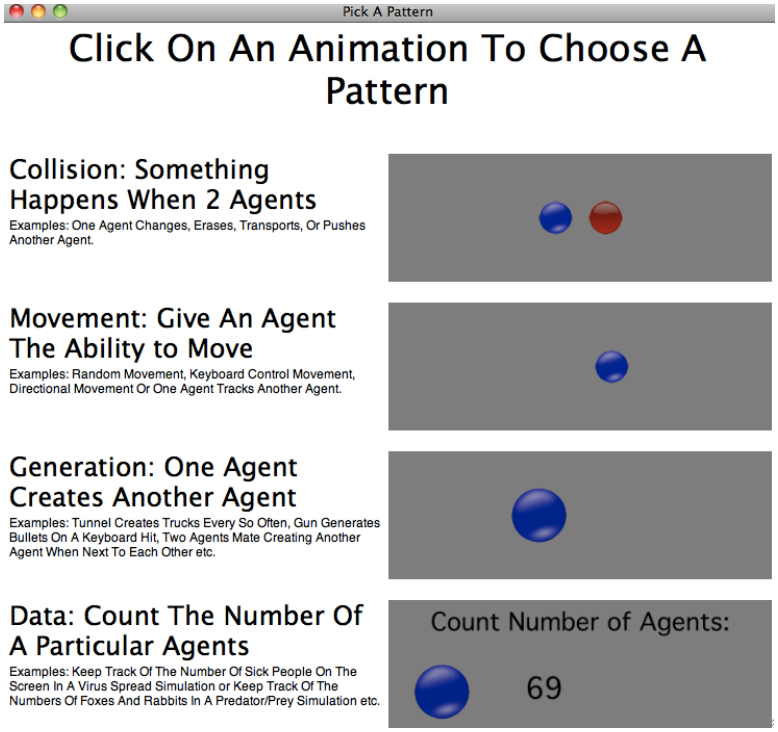
Summary Of Patterns and The Tool:

To start the Tool that will allow you to add behaviors click on the blue circle at the to bar.

This Brings up the following Window (unless it's already open in the background in which case nothing will happen and you have to find it – ie. Move windows around until you see it).



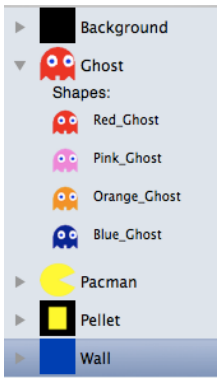
Clicking on the "Add Pattern" Button in the bottom left brings up the Pattern Picker Window. By clicking on any of these animations in this window, you can access all the patterns.



E.2 The Program Descriptions

Pacman Game

Here are the Agents you will be provided with:



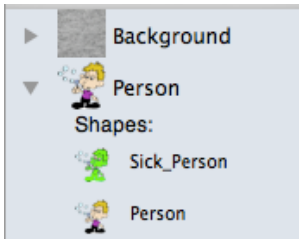
Note that all the agents have 1 depiction except for the Ghost that has 4 depictions (referred to as “shapes” in AgentCubes).

Here is the level you will be provided with:

Here is the description of what you will program; try you best to accomplish the following, only adding behaviors at the pattern level:
“Pacman moves with keyboard keys. All the Ghosts pursue Pacman and when they get to Pacman, Pacman disappears. Pacman eats pellets as he navigates around the level. Neither Pacman nor the Ghosts can go through the blue walls.”

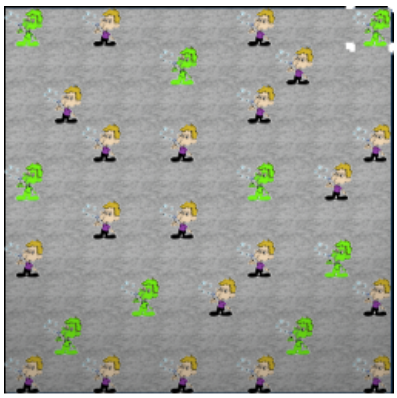
Epidemiology Simulation

Here are the agents you will be provided with:



Note that all the Agents have 1 depiction except for the Person which has 2.

Here is the level you will be provided with:



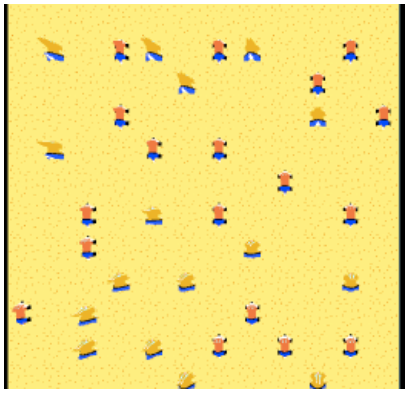
Here is the description of what you will program; try you best to accomplish the following, only adding behaviors at the pattern level:

All depictions of the person move around randomly. A healthy person has a 30% of becoming sick each second that healthy person is next to a sick person. A sick person has a 30% chance of recovery every second. A sick person also has a 10% of death (i.e. disappearing) every second.

Predator/Prey Simulation

Here are the agents you will be provided with:

Here is the level you will provided with:



Here is the description of what you will program; try you best to accomplish the following, only adding behaviors at the pattern level:

Foxes and Rabbits move randomly. Every so often the Fox Agent gets hungry; at this point the Hungry Fox tracks the Rabbit Agent. If the Hungry Fox gets to the Rabbit Agent, it kills it and is no longer hungry. Eventually the dead Rabbit decomposes. Hungry Foxes can also sometimes die of Hunger. Finally, Foxes sometimes reproduce with other Foxes creating a new Fox at some percentage; Rabbits sometimes mate with other Rabbits creating a new Rabbit at some percentage.

E.3 Post Program Creation Feedback Questions

These questions are asked to the participants after their programs were completed:

1) *What pattern(s) in what simulation(s) did you have trouble implementing, if any, and why?*
2) *What would you change or modify about the tool?*

APPENDIX F: Analogical Reasoning Study Participant Answer To Feedback Questions

The following sections contain the six participant's answers to the two feedback questions

F.1 Participant 1

1) What pattern(s) in what simulation(s) did you have trouble implementing, if any, and why?

"I would add a function for an agent to change its own behaviors such as sick agent becoming healthy of its own accord, not interacting with another agent."

2) What would you change or modify about the tool?

"Anytime an agent needed to change its own state or depiction whether through the passing of a certain amount of time, or a percent chance it did not come easily to realize that the only way to do so was through interacting with another agent, even if it's using the same agent."

F.2 Participant 2

1) What pattern(s) in what simulation(s) did you have trouble implementing, if any, and why?

" The slider works pretty funny. Make the depictions larger in the list of patterns. Let s actually change the order of patterns for organizations sake. Add 'self' or a way to tell the user that an agent can trigger something on itself."

2) What would you change or modify about the tool?

"Getting foxes to die. Getting people to die/heal. Having actions triggered by the floor and not by the agents doing the action is non-intuitive. Also had

trouble getting foxes not to die because I had them being polled in all directions, the likelihood skyrocketed."

F.3 Participant 3

1) What pattern(s) in what simulation(s) did you have trouble implementing, if any, and why?

"By limiting the number of interactions available to the agents, the tool actually promoted thinking about how to make agents do what the program needs, despite having no prior knowledge of the parameters of AgentSheets.I think the tool is successful in introducing newcomers to the ideas behind AgentSheets without worrying about the specifies of coding."

2) What would you change or modify about the tool?

"Wall changes person to sick person. I realize that having the sick person act on itself might have been easier, but I'm glad that the tool allowed for roundabout methods of problem solving, counter-intuitive as the may be."

F.4 Participant 4

1) What pattern(s) in what simulation(s) did you have trouble implementing, if any, and why?

" 'Stacked' option could be confusing with wording. Direction options were a little confusing at first."

2) What would you change or modify about the tool?

"Predator Prey simulation: mainly was tough making sure all the patterns were in place and had a bit more complexity with the patterns in general."

F.5 Participant 5

1) What pattern(s) in what simulation(s) did you have trouble implementing, if any, and why?

"Add more features in the collision such as two objects creating another object. The execution from planning is very random."

2) What would you change or modify about the tool?

"I had trouble with the Predator/Prey simulation. There were no provisions as to how to do some of the actions such as decomposing or reproducing."

F.6 Participant 6

1) What pattern(s) in what simulation(s) did you have trouble implementing, if any, and why?

"As agent patterns become more complex, keeping track of what is where is a pain, I would like an improved method of org(anization). Sometimes I think of a better approach and just want to start from scratch, also when things get too "buggy" I would like to do this too. The scroll bar needs to be fixed see you can click up and down its length. Under some agent actions there are tons of options. If one is selected, it should disable the other options that it cannot work with to reduce confusion. If the option is unselected, the other options should become selectable again. This would make setting up the agents more intuitive."

2) What would you change or modify about the tool?

"The directional elements were unclear (drew the directional palette) is only useful in some cases…and it made it difficult when I had a solution I wanted to use and could not find out how to make it work. After I used the system for awhile I became comfortable with the systems limitations, developing solutions became easier."